VOXEL-BASED TOOL SEQUENCE OPTIMIZATION FOR 5-AXIS MACHINING USING HIGH PERFORMANCE COMPUTING

A Thesis
Presented to
The Academic Faculty

by

Amir Ameur

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in the
George W. Woodruff School of Mechanical Engineering

Georgia Institute of Technology
December 2017

VOXEL-BASED TOOL SEQUENCE OPTIMIZATION FOR 5-AXIS MACHINING USING HIGH PERFORMANCE COMPUTING

Approved by:

Dr. Thomas Kurfess, Advisor
School of Mechanical Engineering
Georgia Institute of Technology

Dr. Cristina Tarin
School of Mechanical Engineering
University of Stuttgart

Dr. Christopher Saldana
School of Mechanical Engineering
Georgia Institute of Technology

Dr. Oliver Sawodny
School of Mechanical Engineering
University of Stuttgart

Date Approved: August 10, 2017

ACKNOWLEDGEMENTS

I would like to thank my advisor Dr. Thomas Kurfess for his continuous support and guidance throughout the thesis. His energy and enthusiasm made me feel my work is valuable for the entire research consortium. I would like also to express my gratitude to Dr. Paul Neitzel and Dr. Oliver Sawodny for giving me the opportunity to spend one of the most enriching years in my academic path at Georgia Tech. I owe special thanks to Dr. Tommy Tucker for giving me helpful inputs that improved the quality of my thesis and Dean Cooper for his patience in dealing with my software-related questions. I would like to thank Dr. Christopher Saldana and Dr. Cristina Tarin for their insightful comments and helpfulness. I am grateful to my lab colleagues: Mahmoud Parto, Zhenguo Nie, Roby Lynn, Wafa Louhichi, Dongmin Han, Masoumeh Aminzadeh who have always been motivating and entertaining me in the workplace.

Last but not least, I would like to thank my family for being constantly a source of love, support and inspiration during all these years.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS iii

LIST OF TABLES vi

LIST OF FIGURES vii

LIST OF SYMBOLS AND ABBREVIATIONS ix

SUMMARY x

CHAPTER 1. Introduction 1

CHAPTER 2. Background 5
2.1 Tool Sequence Selection 5
2.2 Tool path generation 12
2.3 Graphics processing unit (GPU) 19
2.4 General-Purpose Graphics Processing Unit (GPGPU) 22
2.5 GPGPU for computer-aided design and manufacturing (CAD/CAM) 25
2.6 Virtualization of Engineering Software and GPU 27

CHAPTER 3. Voxel-Based Cam Software: SculptPrint 30
3.1 Voxelization 31
3.2 Offsetting 32
3.3 Tool Path Generation 32
3.4 Accessibility Maps 33
3.5 Tool Orientation and Retraction 34

CHAPTER 4. Methods 35
4.1 Volume-based Optimization Strategy 37
4.1.1 Greedy Algorithm 37
4.1.2 Application of Greedy Algorithm in volume-based strategy 39
4.2 Average Material Removal Rate (AMRR-based optimization strategy) 44
4.2.1 Solutions for homogenizing the material removal 47
4.2.2 Implementation of the AMRR-based strategy: 49
4.3 Two-Objective Optimization 53
4.4 Experimental Setup 61
4.4.1 Georgia Tech Virtual Laboratory 62
4.4.2 Amazon Web Services 64
4.4.3 Hardware Comparison 65

CHAPTER 5. Results 67
5.1 Results of Volume-Based Optimization Strategy 70
5.1.1 Variation of Tool Sequence with Path resolution 78

5.1.2 Variation of Tool Sequence with Graphic Resolution 83
5.2 Results of AMRR-Based Optimization 85
5.3 Results of Two-Objective Optimization 90

CHAPTER 6. Discussion and conclusions 94

APPENDIX A. 100
A.1 Python Script for Volume-Based Optimization 100
A.2 Python Script for AMRR-Based Optimization 102
A.3 Python Script for Multi-Objective Optimization 105
A.4 Results for Voxel-dependent Step Over 106

References 107

LIST OF TABLES

Table 1: Comparison of tool path planning strategies 16
Table 2: Advantages and disadvantages of volume-based optimization 43
Table 3: Main advantages and disadvantages of the proposed algorithm 51
Table 4: Advantages and Disadvanteges of the two-objective optimization strategy 61
Table 5: Graphic cards specifications 66
Table 6: System specifications for the considered computing platforms 66
Table 7: System specifications for the considered computing platforms 69
Table 8: Example of a tool catalog 70
Table 9: Optimal tool sequence for the candle holder example 71
Table 10: Optimal tool sequence for the tea pot example 76
Table 11: Optimal tool sequence for the pocket example 77
Table 12: Variation of optimal tool sequence with changed step over 79
Table 13: Computation time for tool path generation using T5 with different values of α 82
Table 14: Computation gain for each platform 83
Table 15: Variation of optimal tool sequence with changed voxel size 84
Table 16. Variation of optimal tool sequence with changed voxel size 87
Table 17: Top 6 sequences based on volume removal 92

LIST OF FIGURES

Figure 1: Comparison of accessibility in 3-axis and 5-axis milling 8
Figure 2: Comparison of scallop height for 3-axis and 5-axis machining [18] 8
Figure 3: binary representation of accessible points 11
Figure 4: Example of traced boundaries for each tool's accessible region [18] 12
Figure 5: Tool path generation in milling operations [19] 13
Figure 6: Polyhedral offsetting approaches 18
Figure 7: Interference issues for polyhedral offsetting 19
Figure 8: GPU peak floating point operations per second 21
Figure 9: GPU peak memory bandwidth 22
Figure 10: Types of hypervisors 28
Figure 11: SculptPrint workflow 30
Figure 12: HDT surface representation in SculptPrint 32
Figure 13: Volumetric Offsetting of a simplified human head 32
Figure 14: Tool path generation through ray casting 33
Figure 15: (left) low-resolution map, (right) high-resolution map of a sample point on the generated path 34
Figure 16: Case of failure of greedy algorithm. The red elements represent the choices made by greedy algorithm, whereas the green elements are the ones forming the optimal solution 39
Figure 17: Offseting realized by a Tool i with a less machinable concavity 40
Figure 18: Algorithm's flowchart for volume-based optimization strategy 42
Figure 19: Example of a volume removal curve within a pass 46
Figure 20: Comparison of volume removal before and after filtering 48
Figure 21: Flowchart of AMRR-based strategy 52
Figure 22: Machining time problematic with the greedy problem solving 53
Figure 23: Multitree directed acyclic graph for machining time based optimization 55
Figure 24: Removing non-redundant subsequences 57
Figure 25: Clustering of feasible solutions and selection of the optimal solution 58
Figure 26: Manhattan Tourist Problem 60
Figure 27: Tesla M60 graphic card 62
Figure 28: Georgia Tech Virtual Lab 63
Figure 29: Virtualization hierarchy of SculptPrint [59] 64
Figure 30: AWS data center in India 65
Figure 31: Summary of tooling optimization algorithms 67
Figure 32: Offsetting realized by tools T1 a), T2 b), T3 c), T4 d), T5 e), and T6 f) 72
Figure 33: Removed volume within the first 6 passes and the selection of the optimal tool 73
Figure 34: Removed volume within the first 6 passes and the selection of the optimal tool 74
Figure 35: Volume removal through the optimal tool sequence 74
Figure 36: Cumulative volume removal for the candle holder 74
Figure 37: Shape transofrmation during diffrent stages of machining 75

Figure 38: Dimensions of the 3rd part 77
Figure 39: Path resolution using T1 with (a) $\boldsymbol{\alpha = 0.3}$ and (b) $\boldsymbol{\alpha = 0.7}$ 78
Figure 40: Relationship between SP file size and path resolution..... 80
Figure 41: Benchmarking results..... 80
Figure 42: Impact of voxel size on computation time 85
Figure 43: (a) Filtered and (b) unfiltered path for the candle holder example using a lower threshold of 0.4 86
Figure 44: Volume curves of the Tools (a) T1, (b) T2, (c) T3 during the 1st pass 88
Figure 45: Volume curves of the Tools (a) T3, (b) T4, (c) T5 during the 3rd pass..... 89
Figure 46: Volume/time curve for producing the candle holder..... 90
Figure 47: Simplified two-objective tooling problem 91
Figure 48: Visualization of two-objective optimization results..... 93

LIST OF SYMBOLS AND ABBREVIATIONS

AMRR	Average material removal rate
CUDA	Compute Unified Device Architecture
CPU	Central processing unit
GPU	Graphics processing unit
GPGPU	General purpose GPU
MRR	Material removal rate
MTP	Manhattan tourist problem
SP	SculptPrint
VBO	Volume-based optimization
VM	Virtual machine

SUMMARY

Tool selection is a crucial stage in process planning that involves multiple machining and cost factors and necessitates the integration of CAD and CAM software. Most of the time, this activity requires human intervention through process planning engineers to search adequate cutters in tool catalogues and make decisions based on their experience. With increasing part complexity, more sophisticated CNC machines are implemented and the manual tool selection doesn't generally lead to optimal choices. This thesis presents an approach for tool sequence optimization in the case of 5-axis machining. Most of the reported work suggests tooling optimization methods involving parametric surfaces and CPU-enabled algorithms. In the current work, a novel voxel-based approach is presented. The main advantage of this 3D-representation is the ability to parallelize different operations executed on single voxels and run them on parallel platforms such as GPU cores. This work is realized through *SculptPrint*, a voxelized GPGPU-enabled CAM software, and introduces 3 different algorithms to optimize the tool sequence selection. Each of the formulated strategies is based on the optimization of one or two machining objectives and has a GPU-only implementation. Applications of Cloud manufacturing are also explored via Amazon Web Services and a school-hosted virtual machine by running the developed algorithms on different local and virtualized platforms. The performance of several GPUs is benchmarked and shows an efficiency optimum when using the most powerful hardware. The effects of machining and rendering parameters are investigated. The results show that generated tool sequences are strongly dependent on the chosen step over, voxel size and optimization criterion.

CHAPTER 1. INTRODUCTION

Subtractive manufacturing (SM) or machining is a very old manufacturing concept that has been evolving for a long time. It regroups a variety of processes performed through a controlled material removal from a solid block to reach a desired final shape and size. (SM) contributes to a large number of products manufactured all over the world and constitutes a major component of the manufacturing industry. It has several advantages including high precision, good surface roughness and the ability to treat different materials and produce a wide range of 3D-geometries. From aircraft turbine blades to dental implants and automotive parts, machined products are present in every aspect of our daily life.

Subtractive manufacturing can be challenging and requires experienced labor to be performed correctly. And with growing demand for complex parts and higher productivity to derive competitive advantages, CNC (computer numerical controls) machine tools have become an essential approach to produce machined parts with higher accuracy and efficiency. Throughout the years, CNC multi-axis machining has undergone multiple stages of development: from traditional 3-axis machining where the cutter can move along a fixed axis to every point (X,Y,Z) of its 3D-workplace through 3 translational degrees of freedom (DOF) to 5-axis machining which expands the linear motion in 3-axis machining by adding two rotational joints enabling a wide range of tool orientations and therefore higher accessibility.

G-code is a programming language used by CNC machine tools to control tool motions. It consists of a sequence of commands or instructions through which the cutting tool follows the defined toolpath in order to create the desired workpiece. Generating the

G-code can be done manually or through CAM (computer aided manufacturing) software. The use of these software is particularly crucial in the case of parts requiring machining with many axes such as 5-axis machining, since toolpath planning and orientation can be hard to handle manually and possible to perform only numerically. Based on solid modeling schemes, some CAM software describe solid geometry using boundary representation (B-rep) scheme while others have a voxel-based modeling approach. B-rep is the method adopted by typical CAM software where shapes are represented by their boundaries and each point of the space can be tested against the solid via its boundary. With this approach, analytical models are applied and their complexity and accuracy are strongly dependent on the precision of the computation platform that processes them. On the other hand, the voxel-based representation discretizes the solid volume into small cubes called voxels which are referenced by their (X,Y,Z) coordinates similarly to the 2D case where images are discretized into small pixels and where each pixel has a unique (X,Y) identifier. Voxels are cubes of equal size arranged in a 3D-grid and are treated by the software as spatial arrays (spatial occupancy enumeration) and the accuracy of computational results depend in this case heavily on the resolution of the discretization: the finer the meshing the more accurate the result. This leads to a trade-off between accuracy and computation time.

This work proposes the use of a voxel-based CAM software *SculptPrint* which adopts a hybrid dynamic tree (HDT) to structure voxelized part models. Since voxels are large spatial arrays, different operations can be executed simultaneously to speed up the processing of these arrays. In fact, the simultaneous data processing can be done through parallel computing platforms and an example of that are graphics processing units (GPU)

as replacement of the single thread CPU adopted by common CAM software with parametric surface representation. The implementation of GPU has opened the door to the usage of high performance computing capabilities in digital manufacturing and the acceleration of product development time especially in the case of complex products.

Motivation

One of the major challenges of the manufacturing industry is increasing the productivity by reducing the machining time required to produce parts. This task incorporates a multi-objective optimization targeting different parameters during the process planning such as machining parameters (speed, feed, depth of cut), production size scheduling and tooling. Most of the time, machining operations rely on the user's experience, its cumulative knowledge (vicarious learning) and its subjective evaluation and choices of process parameters. Therefore, a lot of optimization potential can be wasted through the choice of conservative options leading to higher machining time. For this reason, CAM software can offer efficient solutions to optimize machining time by simulating different scenarios and visualizing each time the cutting conditions that may affect the outcome of the planned process. One of the approaches that can be adopted to minimize machining time is the optimization of tool sequence selection during machining. It relies mainly on the idea of multi-cutter machining that substitutes for single-cutter machining and results in generating shorter toolpaths leading to lower machining time. Additionally, modern CNC machine tools offer rapid automatic tool change mechanisms enabling the usage of a muti-cutter set without intervention of the user. Particularly in the case of 5-axis machining where tool motion is more flexible and surfaces are more accessible, conservative machinists tend normally to use the smallest tool most of the time

as it is the tool guaranteeing the highest accessibility, and neglecting the potential of larger tools which can be implemented in different regions of the part.

The goal of this thesis it to automate the process of multi-cutter selection for the production of a complex part using a voxel-based CAM software *SculptPrint* and optimize the tool selection based on different criteria such as volume removal, material removal rate and machining time. The automated process is then simulated on several computing platforms including personal desktop, GT virtual machine (VM) and cloud-based Amazon web services (AWS) in order to perform a benchmarking study aiming at comparing computation time using different (virtualized) GPUs.

Organization of The Thesis

The remainder of the thesis is organized as follows: First, background information is presented on related work on tool sequence optimization especially for the case of 3-axis and 5-axis machining. Second, *SculptPrint* is presented with a focus on the usage of GPU for computational purposes, graphic user interface of the software and the automated scripting via Python. Next, optimization strategies for tool selection are introduced and implemented on a sample (complex) part using different computing platforms and meshing intensities. The results of these strategies are then illustrated, discussed, and compared to the results of a relatively easy-to-machine part where tool selection can be intuitively performed without the intervention of the software. Conclusions evaluate the advantages and limitations of the used methods and give recommendations for future work.

CHAPTER 2. BACKGROUND

In this chapter, the relevant background information for tool sequence selection for milling operations is given, coupled with related work done in the case of 3-axis and 5-axis milling. Furthermore, reported work on tool path generation is introduced- Additionally, examples of GPU utilization in manufacturing applications and acceleration of engineering software are presented. Finally, Virtualization of GPU for computational purposes is introduced.

2.1 Tool Sequence Selection

Tool selection is an essential part of manufacturing process planning because it influences not only the productivity but also the surface topography [1]. Human intervention is still needed in most of the available CAM software to select tools and adequate cutting parameters: a process that involves a trial-and-error method and requires a skillful intervention based on the user's experience and knowledge about manufacturability requirements [2]. However, user's involvement in tool selection process can be limited by an increasing geometric complexity and the presence of freeform surfaces that require simulative capabilities to make optimal choices.

The literature on multi-cutter selection is quite extensive. However, most of the reported work on this topic is focused on 3-axis machining of 2.5D pocket geometries due to the following reasons:

- Pocket geometries are among the most common geometries present in industrial components

- Optimization problem can be reduced to a 2D-graph
- Possibility of the application of graph theory techniques to solve the optimization problem
- Simple tool path generation
- Easy-to-compute machining time

Analytical models were developed by researchers to find the optimal tool set for machining a pocket geometry. Gau and Veeramani [3] developed a dynamic programming-based algorithm for the optimal tool selection in the case of regular convex polygonal pockets with rounded corners. The method decomposes the pocket into subpockets using Voronoi diagrams and assigns a tool to each resulting subpocket. Chen and Fu [4] used a genetic algorithm (GA) optimization model to find the maximum area covered by a cutter and combined it with a bisection method to address pockets bounded with circular arcs and splines. Jonjea *et al.* [5] applied a greedy tool heuristic to rough milling of different layers of pockets surrounded by NURBS surfaces. Nadjakova and Mcmains [6] developed an approach using Djikstra's search algorithm to find an optimal set of cutter radii for machining a 2D pocket within a given machining time that is a percentage of that for the optimal cutter radii sequence. Ramaswami *et al.* [7] present a methodology for optimal tool sequence selection to end mill a concave pocket with or without islands within a minimized machining time using a staircase milling strategy. Mount *et al.* [8]presented a polynomial time approximation algorithm for the multiple-tool milling problem to machine a given region in the plane with minim cost using a given set of tools of different sizes. Offsetting was also another technique adopted to address the multi-tool machining problem. It consists in generating planar offsets on the pocket geometry (outer pocket contours and

inner island contours) using CAD/CAM software. The resulting geometry defines the accessible area of each tool. Lim *et al.* [9] formulated an algorithm for calculating the volume of a 2D-profile, accessible by a given diameter of milling cutter. Based on offsetting results and machining features, they developed a ranking scheme to optimize the multi-cutter selection. D' Souza *et al.* [10]introduced a graph based method to compute accessible and decomposable areas for a pocket geometry using 2D-contour offsetting. The shortest path was calculated using Dijkstra's algorithm, applied in a single source single sink directed acyclic graph representing all possible combinations of tool sequences. The methodology was extended to involve tool holder collision avoidance [11]. To overcome computational limitations present in D' Souza's work, Ahmed *et al.* [12] proposed a genetic algorithm based approach which reduces significantly the computational load required to solve the optimization problem after decomposing the machinable area into subregions using geometric methods. Churchill et al. [13] presented a multi-objective approach for the optimization of rough milling while considering simultaneously the tool sequence, tooling costs, machining time and thickness of excess stock. They used unconstrained NSGA-II as the base algorithm for multi-objective optimization but different preferential search techniques including weighted objective method (WO) and Guided Elitism were adopted to attempt to deal with the problem constraints.

Although the aforementioned approaches are effective in solving the tool sequence optimization problem for 2.5D pocket geometries, they are limited to the case of 3-axis machining and the implemented algorithms are not trivially transferable to 5-axis machining due to the additional rotational degrees of freedom in a 5-axis machine. As shown in Figure 1 a cutter in 5-axis end milling has higher accessibility than in 3-axis end

milling. With a 3-axis machine and a particular cutter-part configuration, only regions that are visible for the cutter in one direction are machinable, since the machine has only linear axes. A reconfiguration of the setup is hence necessary to address inaccessible areas, whereas using a 5 axis-machine allows the machining of these regions by dynamically reorienting the tool to the desired position.

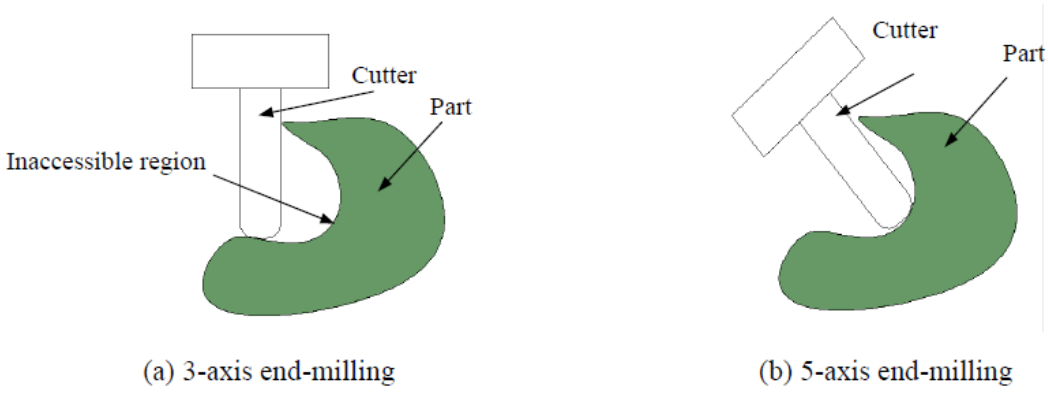


Figure 1: Comparison of accessibility in 3-axis and 5-axis milling

Regarding surface finish, the tool's cutting profile in 5-axis end milling can match closely the surface profile of the machined part by adjusting the tool orientation in such a

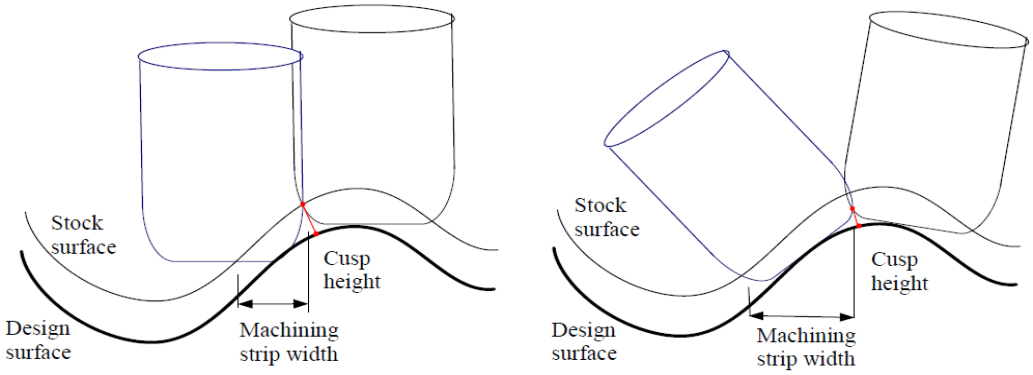


Figure 2: Comparison of scallop height for 3-axis and 5-axis machining [18]

way that much less scallops are left on the surface and the amount of the required hand polishing is significantly reduced.

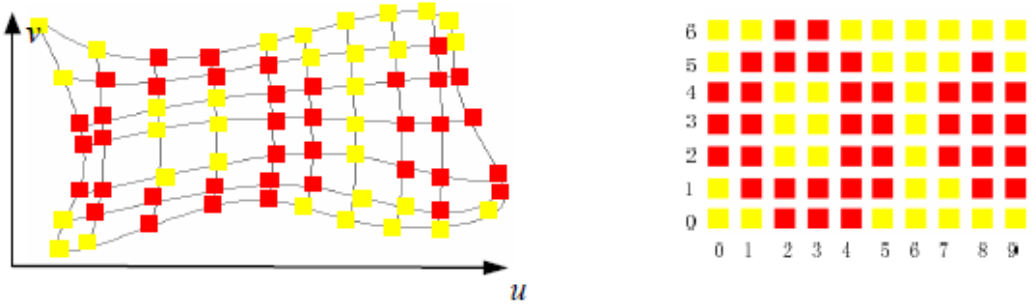
Despite these advantages, the 5 DOF including the two rotational joints, lead to higher complexity in the process planning; in particular tool sequence optimization.

Elber [14] proposed an approach to decompose freeform surfaces, in the framework of 5-axis machining, into 3 categories: convex, concave and saddle-like regions. While flat-end tools are assigned to convex regions, ball-end cutters are suggested for the other regions. A geometric algorithm was used to determine the cutter sizes. On the other hand, reported work on 5-axis surface finishing concentrates on the single-tool selection process. The main idea consists in finding the largest tool that can perform an interference-free toolpath over the totality of the surface. This cutter has normally the highest machining efficiency. Lee and Chang [15] proposed a methodology to select cutters for the 3-, 4-, and 5-axis sculptured surface machining. Their idea is to find the maximum effective cutting radius at every discretized point of the surface. A physical model including tool size and tool orientation angles (incline angle and tilt angle) is formulated and the accessibility cone at each sampled point is constructed. The identification of feasible cutters takes place based on a geometric evaluation of the cutting curvature and surface curvature at a sample point. Jensen *et al.* [16]presented an automatic cutting tool selection methodology for 5-axis machining based on the techniques of curvature matched machining. The proposed algorithm aims at minimizing the machine error and optimizing the material removal rate for the case of a fillet-end mill, chosen from a standard tool catalog. The approach considers also local tool gouging and global tool interference by investigating cutter radius, cutter length and cutter corner radius and possesses a trial-and-error nature. After voxelizing the

target surface, the tool travels along the feeding direction and detects at every sample data point the tool interference with the goal of finding an interference-free orientation. If the algorithm fails in finding an adequate orientation, the next tool in the library is selected and undergoes the same search process.

Within the same context of single-cutter selection, Li [17] developed an algorithm to find the optimal fillet end cutter for 5-axis machining of sculptured surfaces without following the toolpath generation process. In this algorithm, the surface is discretized into sample points with a given resolution that lies within a specific tolerance window that matches real conditions. The accessibility map is constructed for every tool in the library and at every sampled point independently from the machining direction. If the accessibility map is non-empty, then the cutter is considered as feasible and the largest feasible cutter is selected as the optimal one. The optimality of this method is restricted to the case of single cutter selection for the finishing of a whole surface, which does not take advantage of the variety of cutters in the library, especially the larger ones which would lead to higher efficiency. Furthermore, the ranking of cutters is done based on a heuristic that considers cutters with larger major radius as large and if the two cutters have the same major radius, the minor radius is the considered comparison criterion. Haiyan [18] extended this work by developing a heuristic for an automated optimal multi-cutter selection that minimizes machining time and results in an acceptable level of accuracy. The idea is to find the set of cutters that can jointly machine a given surface without interference. The task can be summarized in 3 steps: extracting the feasible (accessible) cutters from the library, setting a threshold for each cutter's accessible region and eventually, formulating a heuristic-based approach to estimate machining efficiency for each cutter. The first step consists in

constructing the accessibility maps for each cutter at every sampled point on the surface profile. Based on the results of this step, 3 types of cutters are identified: accessible cutters that have non-empty A-maps through the whole surface; partially accessible cutters which have both empty and non-empty maps and inaccessible tools which have relatively larger geometries preventing them from being accessible at none of the sampled points of the discretized surface. Next, the feasible cutters (accessible and partially accessible) are ranked based on their sizes where the largest accessible tool is selected as the one to start with. Accessibility regions for each cutter are constructed by tracing a boundary for all accessible points on the surface. Figure 3 shows the accessible points on the considered surface where a binary representation is adopted: red color stands for accessible and yellow for inaccessible. Figure 4 illustrates the traced boundaries and the created accessible regions for each tool. Moreover, a threshold is set for each tool to eliminate relatively small accessible regions compared to tool radius to avoid inefficient cutting and prevent the tool from traveling long distances just to machine a minimal area. The regions below the threshold are then assigned to smaller tools. Finally, since at this stage, the toolpath is not generated yet a heuristic method is applied to estimate the toolpath length based on the calculation of each tool's strip width.



(a) 3D view (b) 2D view

Figure 3: binary representation of accessible points

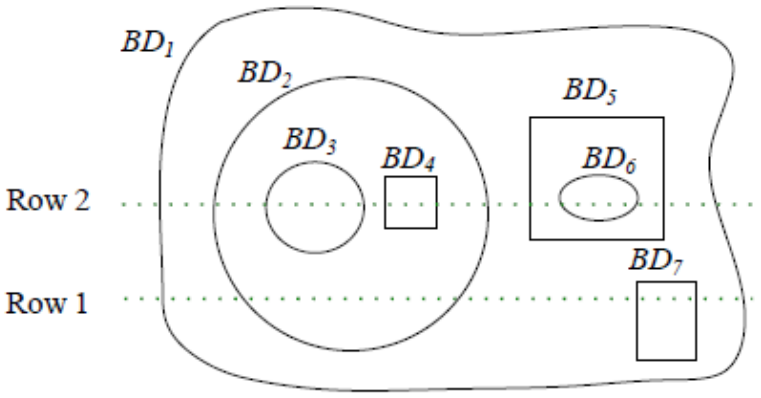


Figure 4: Example of traced boundaries for each tool's accessible region [18]

The tool selection process for five-axis machining is generally coupled with toolpath generation process. Several research papers that deal with the optimization of tool sequence selection consider the automation of toolpath generation and based on the generated toolpath, tool selection can be performed.

2.2 Tool path generation

Tool path planning lies in the heart of CNC machining as it presents the process of transforming a surface into a series of discrete commands, formulated in machine language, that can be fed to a CNC machine in order to achieve the desired movements and produce the real part from the stock. The following figure highlights the importance of path planning in milling operations as a central component of the process planning.

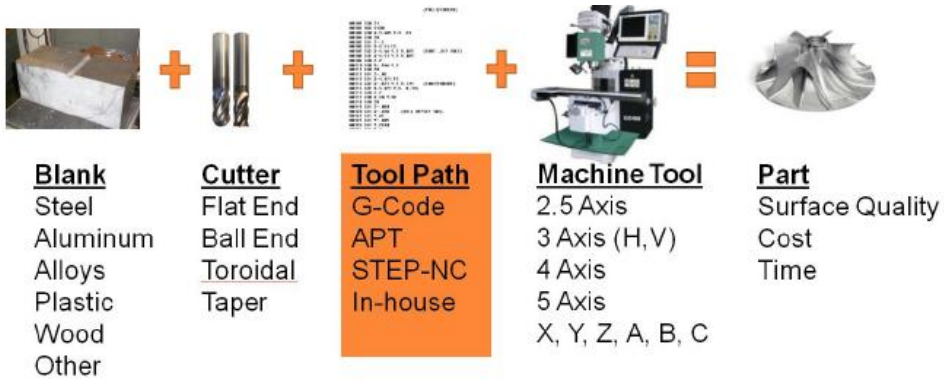


Figure 5: Tool path generation in milling operations [19]

CAD Software offers nowadays a wide range of surface representations enabling different geometric or Boolean operations that can be performed to generate parts with higher surface complexity. After designing the part, CAM software can be implemented to automatically generate toolpath for the designed part models.

The objective of tool path planning is to give accurate positions of the tool center point (TCP) while considering at the same time the overall contact area between the tool and the machined surface to avoid two main problems: gouging and collision.

The traditional approaches for multi-axis tool path planning consist in developing algorithms to generate topological patterns such as serial [20] [2] [21], radial [22] and contour patterns [23]. Other methods take advantage of the parametric surface representation to directly map surface parameters onto Euclidian space. Iso-parametric paths have been first proposed by Ozsoy and Loney [24]. The generated tool path utilizes the data on surface points and places the cutter location point (CL) on the surface normal vector while keeping a distance equal to the cutter radius from each point of the parametrized surface. The technique is performed continually by holding one parameter

constant and incrementing the other until the whole surface is covered and no more steps can be done. Hwang [25] extended this work by generating a parametric offset to the considered surface in order to realize an interference-free toolpath planning. Yuwen *et al.* [26]presented an approach to iso-parametric tool path generation for triangular meshes in polyhedral machining which keeps the path boundary-confined and avoids corners with tight radius that are present in typical paths.

. The main disadvantage of the iso-parametric methods is a non-uniform scallop height distribution when mapping surfaces with large gradients resulting in big gaps between two successive paths in Cartesian space. Elbert and Cohen [27] formulated a fix for this drawback and developed an adaptive method for iso-parametric path planning.

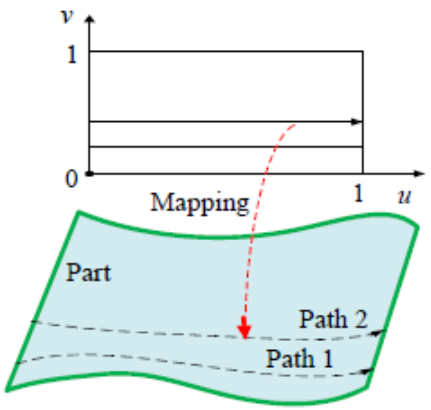
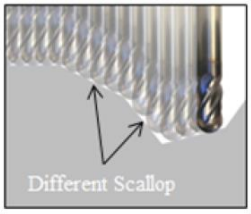
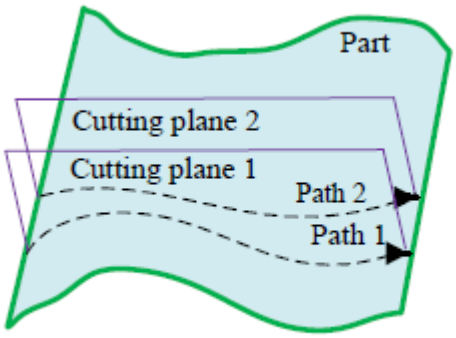
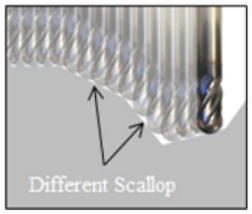
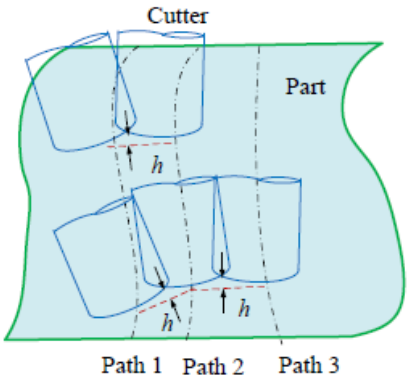
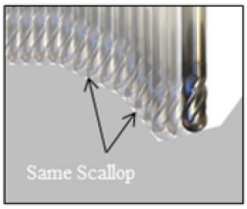
Another widely used method in toolpath generation is iso-planar path planning. This approach consists in generating parametric curves resulting from the intersection of a finite number of parallel planes (constant plane intervals) with the free form surface. The construction of intersection planes is especially useful when the path coincides with one coordinate axis of the tool. For iso-planar topology, it is important to specify the cutting direction (or normal vector of the collinear planes) as this direction has an effective impact on tool path length and therefore machining time: if the planes are vertical, machining is performed with a constant Z-coordinate (Z-level method), whereas the topology with horizontal planes is referred as contour planes offset [28]. Despite its efficiency and reliability to address complex surfaces, this method presents two main drawbacks:1- The high computational load for generating paths for part models possessing a high level of surface complexity, since the computation of intersection points requires solving a system of algebraic equations with boundary conditions; 2- In a very small number of cases, the

iso-planar method would lead to an evenly distributed scallop height. The error is intensified in the case of surfaces with large gradients, where adaptive algorithms [29] should be implemented to compensate this drawback and parametrizes the plane interval, so that a nearly constant scallop height can be achieved.

As alternative tool planning method, iso-scallop path generation is an extension of both iso-planar and iso-parametric methods. To maintain a constant scallop, an adaption based on width between paths is necessary. The adaptive algorithm is of recursive nature: every newly generated path is dependent on the previous one and has to compute find the optimal parameter combination to produce a predefined scallop height. These parameters include surface gradients, tool geometry, posture, and orientation. Related work on iso-scallop tool path generation can be found in [30], [21] and [31].

Table 1 summarizes the aforementioned approaches for tool path planning strategies in terms of path generation method and gives examples of scallop topologies where the two first methods do not lead to evenly distributed scallop heights. In the iso-parametric case, the part is represented as a parametric surface $S(u, v)$ where u and v are two independent parameters. The path is defined by following the direction with constant values of u or v and incrementing continually the other parameter [32]. The process is repeated until the entire surface is traversed. In the Iso-planar case, intermediate parallel planes intersect the part surface and generates the desired tool path, whereas the iso-scallop strategy adapts the tool orientation and interplanar intervals to achieve a constant scallop height.

Table 1: Comparison of tool path planning strategies

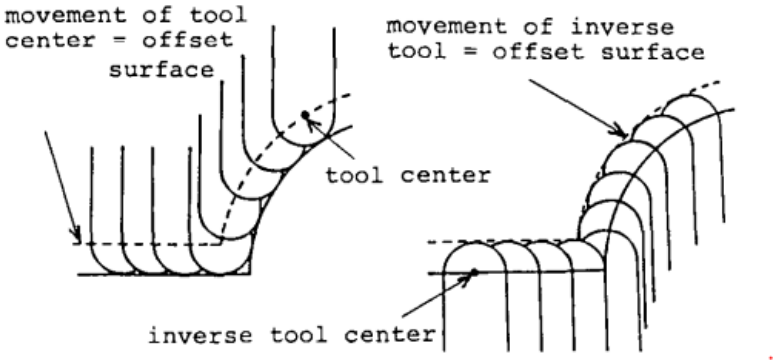
Tool path planning strategy	Graphic representation	Scallop topology
Iso-parametric		
Iso-planar		
Iso-scallop		

The above methods can generally be used with parametric surfaces and require high computational effort for surface checking. A novel method for surface representation and machining was introduced by Duncan [33] and consists in meshing the surface into small interconnected polygons forming a polyhedron. A polyhedral mesh is a set of vertices, edges and faces resulting from a 3D object through a tessellation algorithm. Commercial CAD packages offer tessellation algorithms with a user-defined resolution. The desired accuracy is set based on the tolerance window of the surface and the available memory to store the part. Since the tessellated surface is a polygonal approximation of the true surface, polyhedral machining can be useful in the field of rapid prototyping, where precision is a secondary goal and the rapid creation of a prototype is prioritized [34].

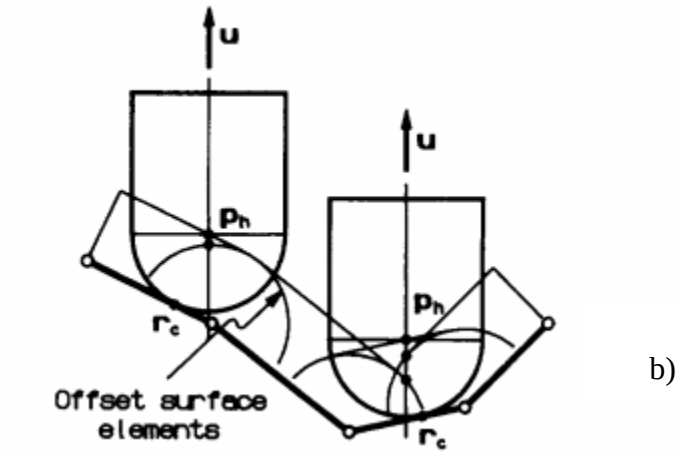
Polyhedral models are stored as STL (Stereolithography) file format which contains a list of the tringles' vertices and their corresponding normal vector. Thanks to this simple representation, STL files are used as universal standard for exchanging 3D meshes between different CAD software. And based on the prototyping process, the optimal level of precision can be determined: for additive processes (rough processes) such as fused deposition modeling, low precision is required, whereas polyhedral machining requires a higher level of accuracy entailing automatically the allocation of larger memory resources, which is no longer an issue thanks to the currently available hardware capacities allowing the storage of large STL files.

Dealing with polyhedral surfaces requires the use of different methods which apply discrete algorithms to detect tool center point and tool contact area, and confront consequently other problems due to the information loss caused by discretization. The tool path generation is realized by offsetting the surface of the triangle mesh using two main approaches: Z-map

and inverse tool offset method. The Z-map approach [25], [35], [33], [36] presents an offsetting technique that consists in dropping the tool along the Z-axis towards the meshed surface until it touches a triangular facet. The z-coordinate is then determined according to the detected height at the intersection point. The inverse tool offset [37], [38] is based on the idea that the movement of tool center inside the envelope generated by the Z-map method lies in the same offset surface created by an inverse tool center movement along the part surface. The tool path generation is executed via an iso-planar approach that slices the offset surfaces and creates the desired. Figure 6 illustrates the two polyhedral machining approaches, whereas Figure 7 highlights the problematic concavities that can



a) Inverse tool offset



b) Z-map method

Figure 6: Polyhedral offsetting approaches

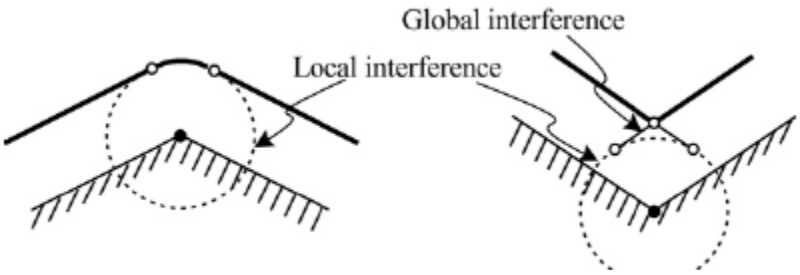


Figure 7: Interference issues for polyhedral offsetting

lead to interference issues (gouging or collision).

To fix interference problems, [35] removes the gouging regions by trimming and reconnecting the offset surface through interpolation. Ren et al. [39] propose a contraction tool method that utilizes a series virtual cutter to search for cleanup boundaries. The generated cleanup path is the union of cleanup curves realized by intermediate virtual cutters, which proves to be a computationally demanding task.

More advanced tool path planning techniques were also successfully deployed in the last 10 years such as C-space based approaches [40], arc intersect methods [41] and the rolling ball [42] [43]. Voxel-based approaches are introduced in the next chapter.

2.3 Graphics processing unit (GPU)

Increasing the performance of computer processors has always been a goal followed by major chip manufacturers. According to processor architecture, there are two main possibilities of leveraging the performance: increasing the clock frequency which is directly correlated with the number of operations per second or increasing the complexity of the circuit i.e. the number of transistors which leads to performing more complicated instructions per clock cycle [44].

Due to physical limitations of processor manufacturing technologies, it is more and more complicated to increase the clock frequency or to add significant number of transistor in a smaller chip surface. For this reason, computational performance growth with these two sources has only been incremental in the last years and needs a substantial breakthrough. Therefore, it is important to find ways of a more efficient usage of available resources.

In general, resolving a problem requires the processor developer to answer a dilemma with two extreme ways: the processor should execute one very complicated instruction per clock cycle; or the processor should perform many simple commands per clock cycle. In the real world, two types of processors are suitable for this tradeoff: Central Processing Unit (CPU) which can perform few complicated instructions per clock cycle and some simple commands, and Graphics Processing Units (GPU) which can perform a large number of simple commands per clock cycle. These processors are originally used in two different fields and deployed to perform different types of operation: while CPU performs most operations in modern computers, GPU is used for 3D graphics calculation. However, in the last years, new generations of these two processors are getting closer from an architectural aspect, since CPU can perform now more and more complicated commands per clock cycle and GPU increased the ability to handle a higher command complexity with some limitations. Consequently, more algorithms with complicated commands can be executed using CPU and achieve good performance without providing many commands per clock cycle, whereas GPU presents an opposite situation, where only algorithms with only simple commands and the ability to parallelize theses commands can meet high performance objectives. In Addition, GPU can run these simple algorithms much faster

than CPU since it is designed for 3D graphics calculations that require independent mathematical operations without having to use the result of other calculations, which is commonly the case of compute-intensive operations such as dynamic simulations, engineering application and video processing. To sum up, GPU is a special purpose CPU, intended to run simple instructions over a large block of data in many parallel streams. Although these tasks are simple and specialized, it is important to know that GPU can be hundreds of times faster at executing these operations than CPU. The trend shown in Figure 8 confirms that GPUs are continuously extending their performance and surpassing single thread CPUs in terms of floating point operations per second. For instance, Intel Core i7-6950x (one of the most advanced commercial CPUs) can perform up to 300 GFLOPS ($3x10^{12}$FLOPS) while nVidia Tesla M60 can perform more than 6000 GFLOPS (20 times faster).

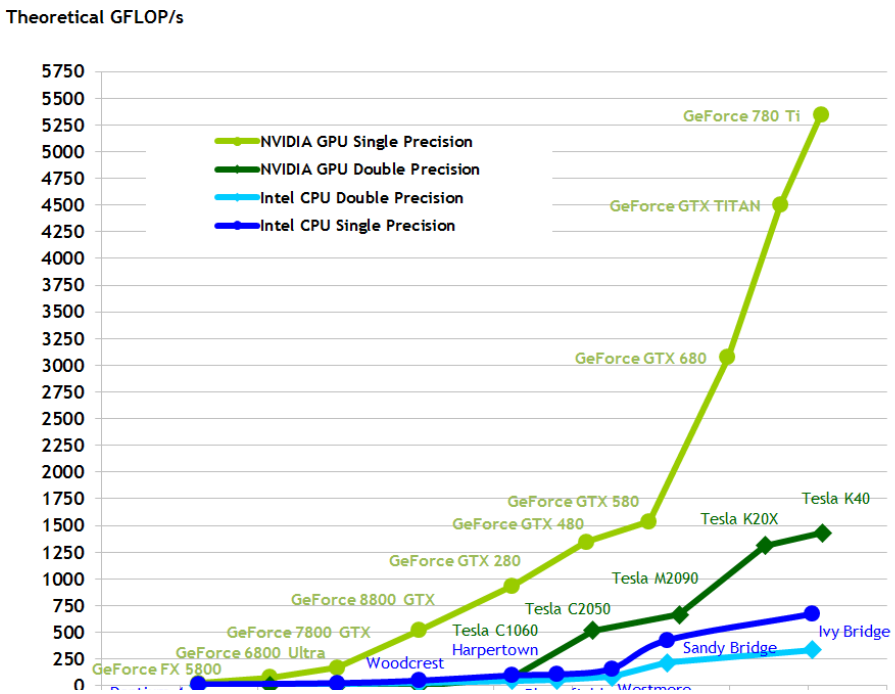


Figure 8: GPU peak floating point operations per second

Memory bandwidth of GPUs has also been increasing in the last years as illustrated in Figure 9: A trend confirming ‘Moore’s law’ that forecasted the computation power would double every 18 months.

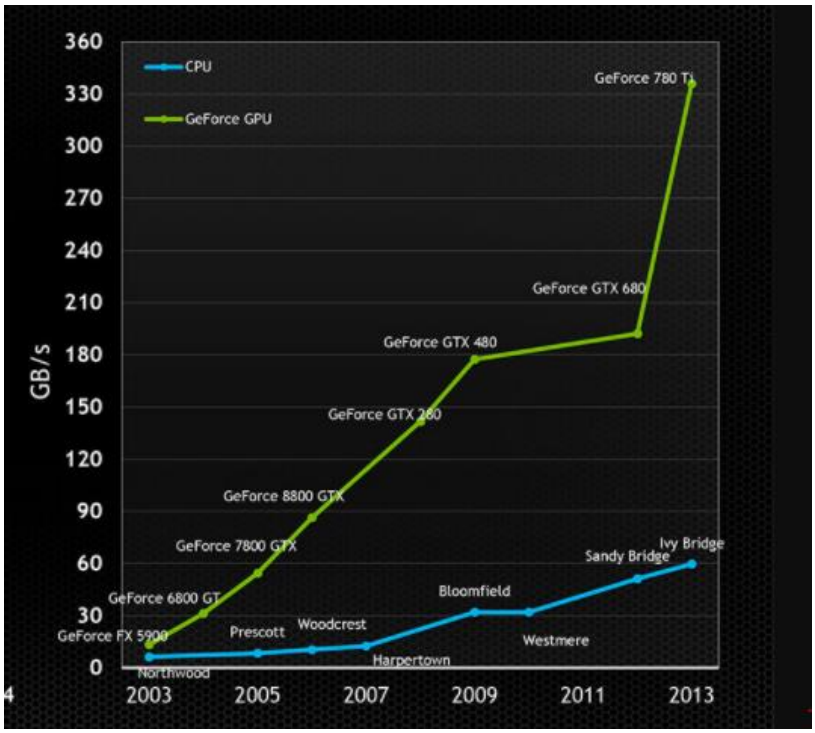


Figure 9: GPU peak memory bandwidth

2.4 General-Purpose Graphics Processing Unit (GPGPU)

GPGPU presents a solution to boost performance of general purpose applications using GPU technology. Since 2005, graphic cards manufacturers recognized the potential of GPU that could be implemented in leveraging high-performance computing and opened a pipeline allowing the users to benefit from a portion of the graphic card to serve general purpose calculations traditionally handled by central processing unit CPU [45]. With this granted capability, research proliferated in GPGPU computing to cover the fields of engineering, medicine, finance, and physics [46]. Thanks to the high number of floating

point operations per second granted by easily embeddable graphic cards, Simulations that were too slow to be executed on CPU became realistic tasks that could be achieved within minutes. In the same context, researchers started investigating ways of paralleling conventional numerical tasks to make them GPU. Galoppo *et al.* [47] formulated an algorithm for solving dense linear systems on graphics hardware by performing LU decomposition using a series of rasterization operations on the GPU architecture and appropriate data structure. Rumpf [48] used GPU to accelerate image processing techniques such as implicit active contours to segment digital images. In this paper, vectors are considered as images and linear algebraic operations are executed by graphics application of image compositing. Haider *et al.* [49] developed a GPU-only Cholesky factorization by removing the expensive CPU-GPU communication. The proposed algorithm outperforms the performance of multi-core CPUs and is 7-8 times faster. In engineering applications, GPGPU has been applied in Finite Element Analysis (FEA) and resulted in a significant computation time savings. The idea consists in allocating parallelizable operations of FEA to GPU and co-operate with CPU (x86) to carry out serial instructions. The focus of parallelization is laid on linear equation solver and reducing the number of rendering passes and an implementation was successful in commercial FEA software such as Abaqus [50] . Other application of GPU in engineering software mainly CAM software are introduced in the next sections.

A common attribute of the above-mentioned applications is the existence of lots of data on which the same computational operations are being executed. Furthermore, interdependency between data elements should be minimal or nonexistent., that implies GPU-friendly algorithms do not use the results of previous commands. Parallelization

requires also the saturation of many arithmetic logic units (ALU) . An implication of these purposed tasks is that programmers need to design applications that are specifically GPU-oriented and possess a high parallelization degree. This requires also new programming languages, processing platforms, data structures and paradigms for parallel processing.

The need for highly parallel algorithms is a prerequisite for a useful implementation of GPGPU. This includes also different work scheduling routines and the reformulation of computational problems in the design phase. For this purpose, GPGPU programming platforms have been developed and offered a framework to harness the power of GPU. The major GPGPU platforms that are available on the market today are: Microsoft Direct Compute, OpenCL from khronos Group [51] and nVidia CUDA [52].

Compute Unified Device Architecture (CUDA) is a parallel computing platform and programming model invented by nVIDIA [52]. It enables developers to use GPU for general purpose computing and presents a software layer offering the access to GPU pipeline and instruction libraries. For the optimal use of CUDA, nVIDIA developed a software toolkit (CUDA Toolkit) that provides a development ecosystem enabling the creation of high-performance GPU-accelerated applications. Additionally, CUDA toolkit includes a compiler, debugging tools and numerical libraries that are compatible with C/C++ development environment. [52] and through API extensions parallelized compute Kernels can be executed.

The acceleration of computing applications achieved by GPU and the speed up advantages compared to CPU use have had significant impact on manufacturing research as discussed in the next section.

2.5 GPGPU for computer-aided design and manufacturing (CAD/CAM)

It is clear throughout the literature (see 2.1 and 2.2) that the simulation and computer based optimization of machining operations has a major limitation which is the computation time required to execute the proposed algorithm. GPGPU functionality can be a solution to accelerate the processing and has generated promising results in manufacturing environment.

One of the main fields in which GPU can be implemented is image synthesis or rendering since CAD packages use this technique to represent the light integration with 3D models. The GPU is specially used to compute the color value of the pixels on the screen resulting from the light exposure of triangle mesh constituting polyhedral models. Another feature that can take advantage of GPU is the extraction of Z-buffer values which are the equivalent of a height map or an image that stores the distances between each point of a 3D graphic and the human eye. The same technique is used to determine object contacts and avoid draw calls of objects that are hidden by other objects having a smaller depth value and are consequently closer to the screen.

Mcmains *et al.* [53] investigated the Z-buffer techniques and developed a GPU-accelerated algorithm to test the castabiliy of geometric parts and examines redesign possibilities. The algorithm is based on the graphical detection of molding undercuts and decides whether a tessellated model can be casted in 2-part mold. This involves a depth buffer functionality that determines the visibility of an object from the mold removal direction, identifies which pairs of facets are interacting and removes those which are not obstructing each other from further processing. The implementation took place on Quadro

FC 3000 series and achieved promising results in terms of computation time. Gray *et al.* [42] presented a GPU-based approach for 5-axis surface machining called Rolling Ball Method which is a tool positioning strategy using the Z-buffer technique to generate an interference free curvature matched tool position. In the same context, the method eliminated the need for surface parametrization and showed its robustness in parallelizing surface checking operations since they are repeated independently on discrete triangulated data sets and regardless of the type of the surface. Krishnamurti [54] formulated new parallel GPU algorithms to accelerate basic CAD functionalities such as surface intersections, orthogonality checking, spline evaluation which improved the quality of the CAD-User environment and the interactivity level. The study dealt with a critical operation of CAD systems which is the evaluation of NURBS surface. The developed parallel algorithm allowed a faster rendering of NURBS and the acceleration of surface-surface intersection algorithms to 50 times faster than the kernels available in commercial Solidworks packages. Moreover, the parallelization of minimum distance computations allowed a high-speed performance that is 200 times faster than the CPU implementation. Roth and Ismail [55]developed a mechanistic model of the milling process based on an adaptive depth buffer in order to calculate cutting forces during multi-axis machining. Tukora and Szalay [56] presented, within the same framework, a GPGPU accelerated algorithm for real-time determination of cutting force coefficients. Hsieh and Chih [57] applied graphics processing unit to solve the optimization problem of machine error estimation for a 5-axis tool path planning strategy. A particle swarm optimization scheme was developed to populate cutter locations with minimum error on machine surface. Innui [38] presented a GPU-assisted algorithm to generate cutter paths for CNC machines and

developed a CAM software based on this technology that is implemented in Mazda corporation for the manufacturing of stamping dies.

All of the above-mentioned researches highlight the importance of GPU as major enabler of high performance computing in the field of manufacturing.

2.6 Virtualization of Engineering Software and GPU

Due to technology limitations, the idea of virtualizing Engineering software such as CAD/CAM has faced a lot of implementation obstacles and was until recently not deliberately investigated. Achievements made in network utilization and GPU technology for resource sharing and virtualized access have enabled the organizations to successfully export their graphics applications to virtual environments.

Virtualization of Engineering Software can generate many benefits for the organization such as leveraging global talents for collaborative design [58] and the ability to use and present design models on mobile devices. In addition, virtualization can contribute to lowering development costs and the time to market by offering a collaboration platform to handle very large design models by a group of engineers based in different geographical locations without having to transfer huge bulk of data, which is generally a slow process.

Hardware limitations can be bypassed through virtualization since it enables the utilization of hardware-accelerated software without requiring an integration of powerful hardware in the personal machine. In addition, it eliminates the restrictions imposed by the operating system configuration and the need for installing specific drivers by duplicating

the display of the virtual desktop for each user and running the software in the cloud independently from the personal machine and the system requirements [59]. A private cloud with a restricted number of users can also be implemented to increase security level and circumscribe the impact of system breakdowns and troubleshooting measures.

Running a virtual machine requires the existence of a monitor referred to as hypervisor on the host machine. The hypervisor is responsible for sharing available hardware resources among the different guest machines and presents an abstraction layer between the underlying physical compute resources and the operating system of the virtual machine. According to Popek and Goldberg [60] there are two types of hypervisors: type 1 called native or bare metal in which the hypervisor communicates directly with the underlying server hardware and the levels of abstractions are minimal; type 2 is referred to as hosted hypervisor where the software is not installed directly on the bare metal but embedded into an already existent operating system as illustrated in Figure 10.

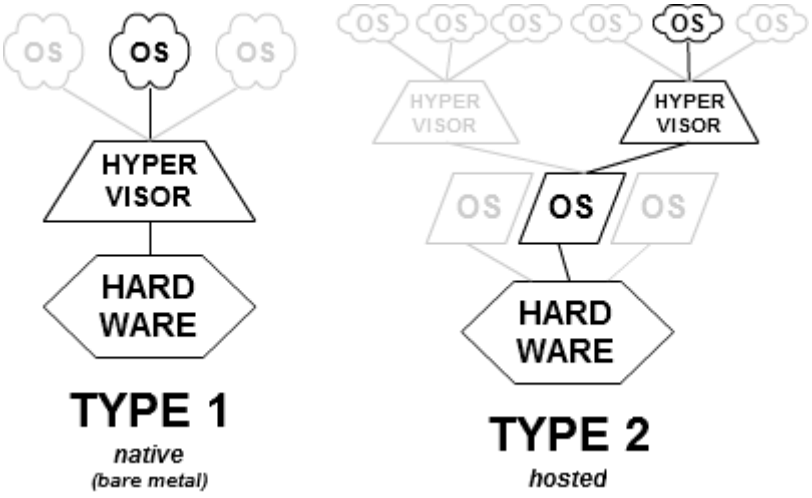


Figure 10: Types of hypervisors

Hypervisors of type 1 can be provided by CitrusXen, Hyper V and VMWare and guarantee direct access to the hardware however their performance is influenced by the number of users sharing the physical resources, which would lead to a system slowdown if the access is granted to a large number of users. For this reason, powerful hardware should be considered to overcome performance bottlenecks.

In the same context, the virtualization of GPU is an emerging field that allows the abstraction of hardware-accelerated applications and the execution of parallel algorithms on virtual machines. Shi *et al.* [61] introduced vCUDA (virtual CUDA) that allows GPGPU computation on virtual machines. The main idea consists in intercepting and redirecting API calls. Gupta *et al.* [62] designed a system for virtualizing and managing GPU without having direct access to accelerator but through forwarding GPU calls via a management layer. Xiao, et al demonstrated VOCL, an OpenCL-based virtualization allows for GPU hardware sharing. The virtualization approach in this thesis is the one adopted by *Lynn et al.* [59] and involves a windows-based implementation of virtual GPU allowing multiple virtual machines to share the same GPU. The system architecture and specifications is introduced in the next chapters.

CHAPTER 3. VOXEL-BASED CAM SOFTWARE: SCULPTPRINT

This work explores the use of a voxel-based CAM software, known as *SculptPrint*, to solve manufacturing-related optimization problems. To produce a given part, *SculptPrint* uses a well-defined workflow that guides the user step-by-step to generate a tool path capable of transforming the initial material stock into the desired part. An overview of the workflow is given by the following Figure:

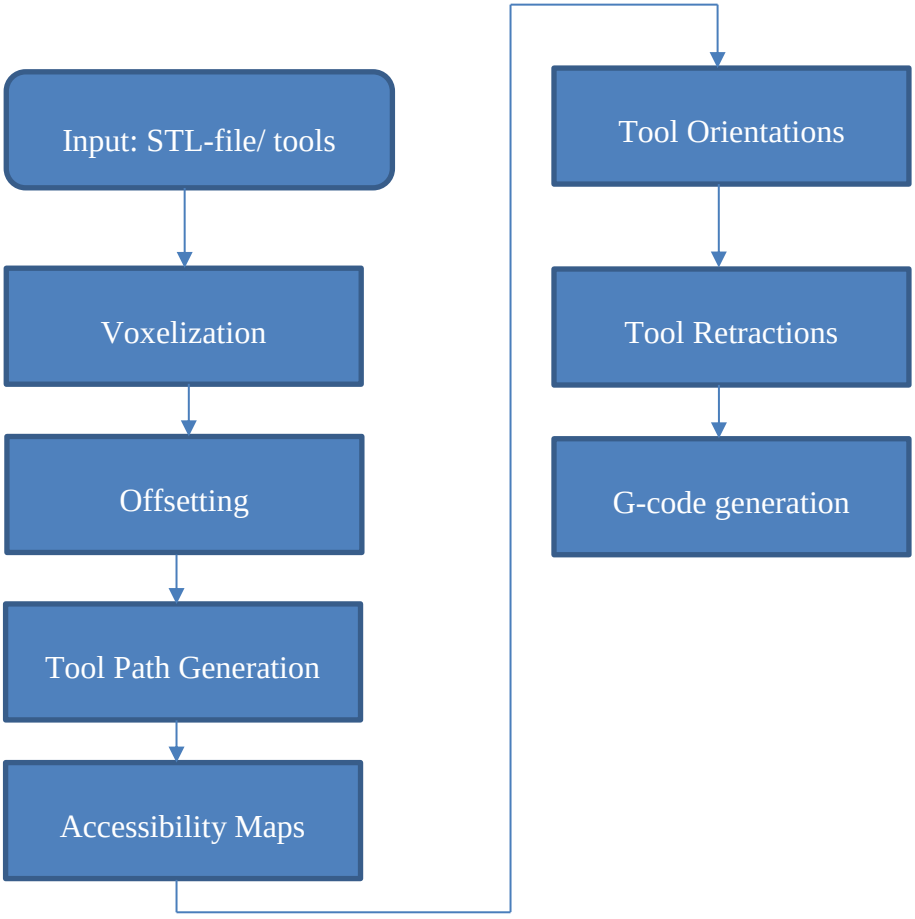


Figure 11: SculptPrint workflow

In this chapter, a brief explanation of the different workflow stage is given.

3.1 Voxelization

Traditionally, CAM software use the boundary representation approach to model solids. To represent a shape, a parametric boundary between the empty space and material is modeled using NURBS surfaces (Non-uniform rational B-spline). Both geometric and topological information are stored in B-rep models; this includes nodes, edges, and faces and most importantly the parametric representation of the boundary.

If all faces of the model are triangular, a polyhedral mesh is formed. This representation is more convenient for graphic rendering techniques.

CSG representation generates solids using simple Boolean operations on basic shapes, generally represented through B-rep.

The voxel-based modeling is a volumetric approach that transforms the volume occupied by a solid into a finite number of cubic units called ‘voxels’, which represent the 3D analogous of 2D pixels. If the discretization is done on regular basis, the volumetric model can be stored in 3D arrays. Otherwise, with irregular subdivisions, a different data structure should be used. *SculptPrint* uses a data structure known as hybrid dynamic tree (HDT) that combines the grid representation and octree structure [63]. Figure 12 illustrates the hybrid structure of the voxelized part. This approach consists in adjusting dynamically the resolution of discretization; the voxels located on the boundary surface have the highest resolution, and the voxelization becomes coarser the farther we go from the boundary. This approach is beneficial for efficient data storage and allows an accurate approximation of complex surfaces, provided the voxel size fulfills the resolution requirements.

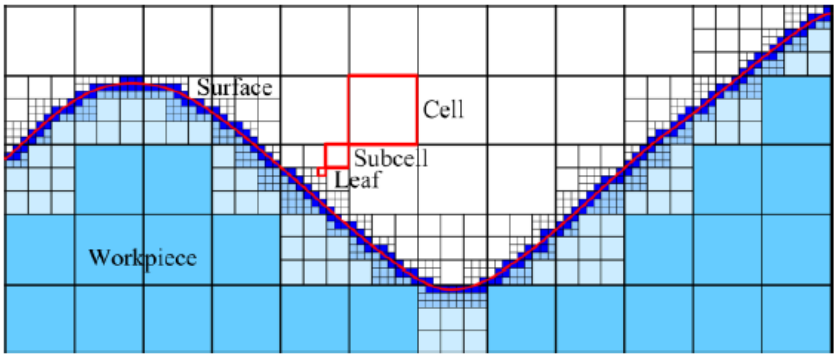


Figure 12: HDT surface representation in SculptPrint

3.2 Offsetting

The offsetting algorithm using HDT was developed by Hossain, *et al.* [64], and presents an image filter based offsetting algorithm which enables the GPU acceleration, and decomposes large distance offsets into successive offsets with smaller distances to leverage memory efficiency. Important manufacturing information such as depth of cut and target volume are used as input parameters for offsetting. An example of surface offsetting is exhibited in Figure 13, where the offsetting is realized with a distance corresponding to tool radius.

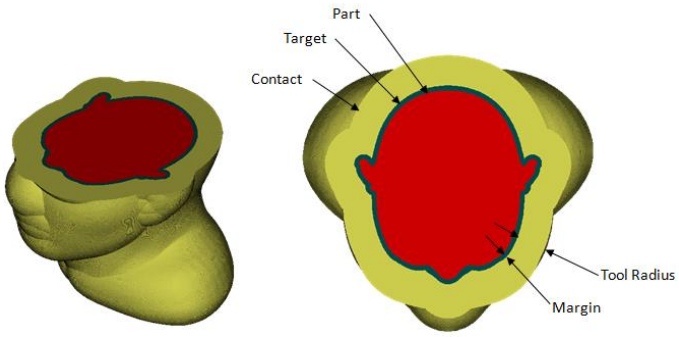


Figure 13: Volumetric Offsetting of a simplified human head

3.3 Tool Path Generation

This step is realized with high automation degree. A gouge-free voxel-based tool path generation process was developed by Tarbutton *et al.* [65]. The idea consists in obtaining digital surface information through parallel ray casting. The gouge avoidance is guaranteed through a voxel-based comparison of tool geometry and boundary voxels. By choosing the machining step over, the path is generated following circular patterns starting from the point with the highest Z-coordinate and checking continuously the intersection of the traced rays with the contact volume, as exhibited in Figure 14.

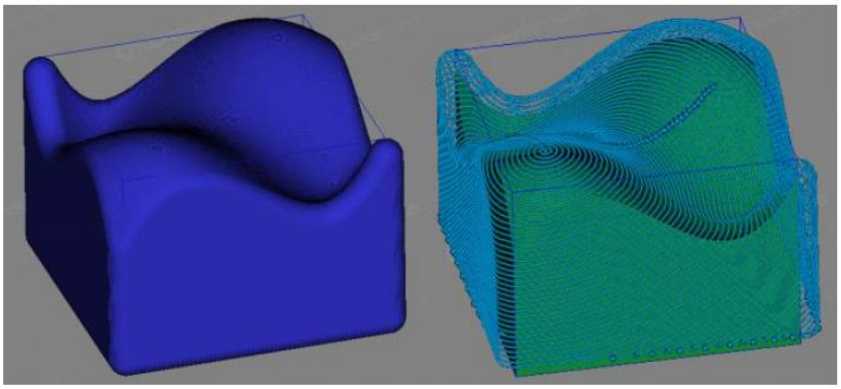


Figure 14: Tool path generation through ray casting

3.4 Accessibility Maps

After generating the tool path, the software checks all feasible tool orientations on each point of the path. This step consists in generating all possible angular combinations of the two rotational joints, present in 5-axis machines to ensure an interference-free path. The resolution of the maps can be modified based on the part's geometric complexity. A comparison between low and high-resolution maps is exhibited in Figure 15, where the ranges of φ and θ are respectively plotted on the vertical and horizontal axis.

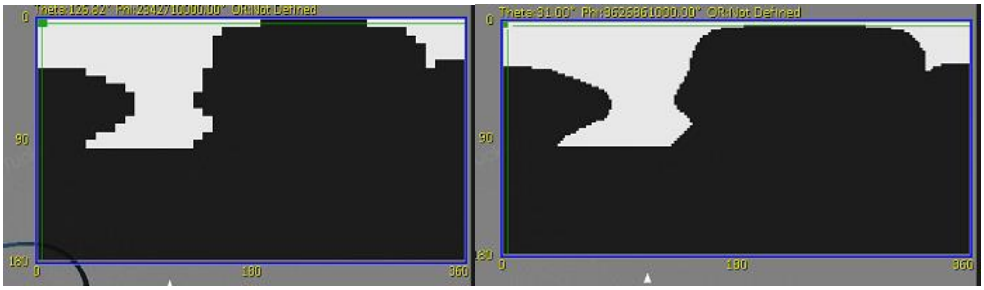


Figure 15: (left) low-resolution map, (right) high-resolution map of a sample point on the generated path

3.5 Tool Orientation and Retraction

The tool orientation strategy consists in selecting an angular combination of the accessible range generated in the previous steps and iterating this process in each point of the tool path. The goal is to minimize the required reorientations and maintain the same tool posture as long as the geometry allows. The tool retraction takes place if an angular threshold is surpassed and is realized through G0 motions; that means with highest possible feed.

CHAPTER 4. METHODS

The tool sequence selection lies in the heart of subtractive manufacturing operations since it has a (direct) tangible impact on the machining time, productivity, and surface finish. As described in the previous chapters, there is only limited work on tool selection for 5-axis machining and the formulated approaches work exclusively with NURBS surfaces and require analytical methods that are based on the analog description of workpiece and cutter geometry. This implies that the GPU-accelerated execution of the presented algorithms is limited since they involve only complex commands that can be processed solely through conventional single thread CPU. Additionally, the related works lay the focus most of the time on surface finishing phase and do not consider the roughing operations that take place in the early stages of the manufactured product being transformed from a stock into a final shape with desired curvature and dimensions. Moreover, the computation of machining time is based on toolpath length estimation and do not involve accurate path metrics such as swept volume and the time required for tool reorientations and retractions.

The use of a voxel-based CAM software would allow the parallelization of multiple operations and a more accurate calculation of path metrics since it is executed along a 3D-array which supports simple commands such as addition and subtraction and whose convergence is dependent on the voxelization resolution. Based on these metrics, optimization algorithms can be formulated to determine the optimal tool sequence from a given tool catalog to produce a specific part. The algorithms should be automated using

the python commands that exist within the python package of SculptPrint at the time of the project realization and have to adopt the same workflow predefined by SculptPrint to create a pass as described in the previous chapter. Depending on the optimization objectives different steps of SculptPrint workflow are performed and their outputs are used to decide upon the optimality of the selected tools.

Based on the constraints and degrees of freedom offered by SculptPrint, the following assumptions for the proposed work are made:

- Only one tool can be used in each single pass
- It is impossible to modify the geometry generated toolpath, since it is an automated capability offered by the python scripting toolkit
- Toolpath resolution can be modified by varying the step over
- All optimization techniques presume the existence of a toolpath upon which the path metrics can be determined and used as optimization parameters
- The used cutters are ball-end mills defined by their diameters and lengths
- The described workflow of SculptPrint should be respected and the optimization algorithm decides until which stage of the workflow the computation is needed
- The use of the available API python capability

In this work, the tool selection covers both roughing and finishing operations. For this reason, it is important to select optimization criteria in such a way that volumetric and/or time aspects are considered. As first candidate, the removed volume can be selected. To

additionally involve machining time, average material removal rate per pass can be selected as second candidate. Finally, a machining time-based strategy is presented.

In this chapter, the optimization strategies for tool sequence selection are introduced with their advantages and disadvantages. Additionally, the experimental setup including cloud platforms for GPU virtualization is presented.

4.1 Volume-based Optimization Strategy

The volume-based strategy is based on the idea of maximizing volume removal per pass. It consists in generating passes with different tools and picking each time the tool that eliminates most of the volume i.e. contributes more towards reaching the target part. This operation is continuously repeated until a percentage of completion is achieved or a maximum number of passes is realized Consequently, the tool selection is ruled by the effectiveness of achieving a defined objective rather than the efficiency of fulfilling it.

The proposed technique can be referred to as greedy algorithm.

4.1.1 Greedy Algorithm

Greedy algorithm is a paradigm that follows the heuristic of finding a local optimum at each stage, with the hope that this choice will lead to a globally optimal solution [66]. For a given objective function that needs to be optimized, a greedy algorithm makes at each stage greedy choices based on a selection function and outputs at the end the elements of the optimal set for the objective function. This algorithm has only one shot at each step and the choices that are made are irreversible. For the implementation, the algorithm requires

a candidate set, an objective function to be optimized, a selection function and a solution function.

Greedy algorithm ($Stop\_condition, X_1,\ X_2, X_3 \dots,\ X_i \dots,\ X_n$)

$S \leftarrow \emptyset$

$j \leftarrow 1$

While Stop_condition is false

$k \leftarrow\ \varphi(X_j)$

$S \leftarrow S \cup \{k\}$

$j \leftarrow j + 1$

Return S

Algorithm 1: Greedy Algorithm structure

The above-introduced pseudocode (Algorithm 1) presents the structure of the greedy algorithm where X_j is the set of possible choices at the stage j, $\varphi(X_j)$ is a selection function applied to the set X_j. k the optimal choice and S the set of optimal choices.

Greedy algorithms lead to globally optimal solution if two properties are valid [67]: greedy choice property: selecting a local optimum will lead to global optimum; optimal substructure: optimal solution to subproblems is element of the optimal solution to the problem.

These properties can be found in problems such as coin changing where the cashier, at each iteration, adds the coin of the largest value that does not surpasses the amount to be paid back or interval scheduling problems where earliest finish time is prioritized.

Greedy algorithm can be advantageous if the problem solution needs to be approximated in a reasonable time since analyzing the run time of such algorithms is relatively easy. However, the short-sightedness and lack of global analysis are the main weaknesses of this algorithm. Therefore, it is obvious that greedy technique cannot be applied efficiently in decision trees and problems involving the recursive investigation of optimality such as travelling salesman problem. Figure 16 illustrates a failure case of this algorithm where at each step the biggest number is chosen. The proposed solution has a sum of 7+12+6=25 while the optimal solution is 7+3+99=109!

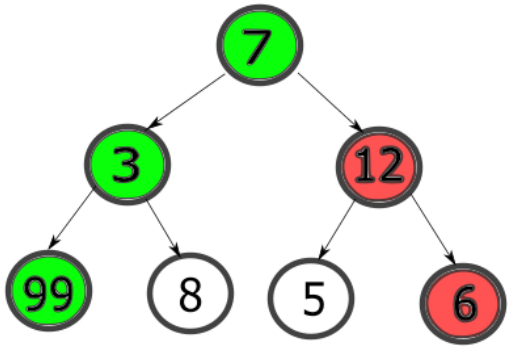


Figure 16: Case of failure of greedy algorithm. The red elements represent the choices made by greedy algorithm, whereas the green elements are the ones forming the optimal solution

4.1.2 Application of Greedy Algorithm in volume-based strategy

The volume-based strategy is inspired by the idea of the coin changing problem. In fact, the formulated algorithm should pick at each stage the tool removing the largest volume among all tools existing in the library. The operation is iterated until a target volume is achieved. For a realistic applicability, once a tool is selected, the algorithm can pick in the

next steps only among the tools that are as large as the selected tool or smaller. Since the considered tools have a ball-end mill, the size criterion refers to the cutting ball's diameter. By using this technique, the shift to a smaller tool takes place after the potential of larger tools is exhausted and smaller tools can consequently machine the regions that are inaccessible for larger tools due to interference problems.

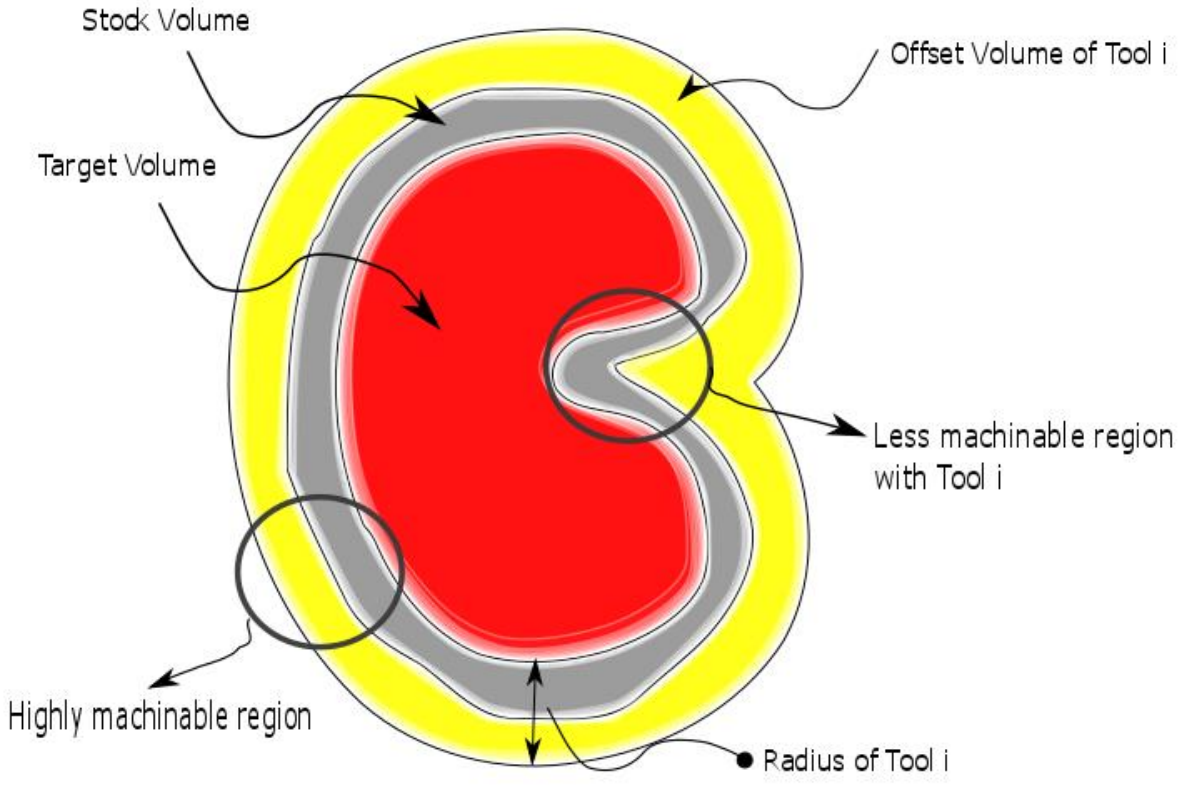


Figure 17: Offseting realized by a Tool i with a less machinable concavity

Figure 17 exhibits a case where the potential of larger tools can be fully used in convex areas and concavities are no more accessible. In this case, the shift to smaller tools for the next passes is a reasonable decision to achieve higher volume removal.

After voxelizing the part, the algorithm can be implemented within SculptPrint as follows:

1. Pick the largest tool in the library
2. Offset the part with the corresponding maximum depth of cut
3. Generate an automated toolpath with a given step over
4. Compute removed volume
5. Try the next largest tool and repeat the same steps 2-4
6. After trying out all tools, pick the tool with highest volume removal
7. Repeat the steps 1-7 until a volume threshold is achieved

The flowchart of the algorithm is displayed in Figure 18 and it contains a loop for trying out all tools and a loop for evaluating the selection function returning the tool with the highest volume removal.

To compute the material removal, an accurate estimation can be directly done from the voxel model by summing the voxel values before and after realizing a pass and subtracting the former sum from the latter. The voxel value is a representation for the portion of voxel filled with material and is ranging from 0-255 [19]. It considers the fact that, in some regions, the tool is not engaged with the total depth of cut. Equation (1) gives the volumetric amount of material V_M in a voxel while the total volume removal by a tool pass is given by Equation (2):

$$V_M = \frac{VoxelValue}{255} \cdot VoxelSize^3 \tag{1}$$

$$V_{removed} = \sum V_{M,before} - \sum V_{M,after} \tag{2}$$

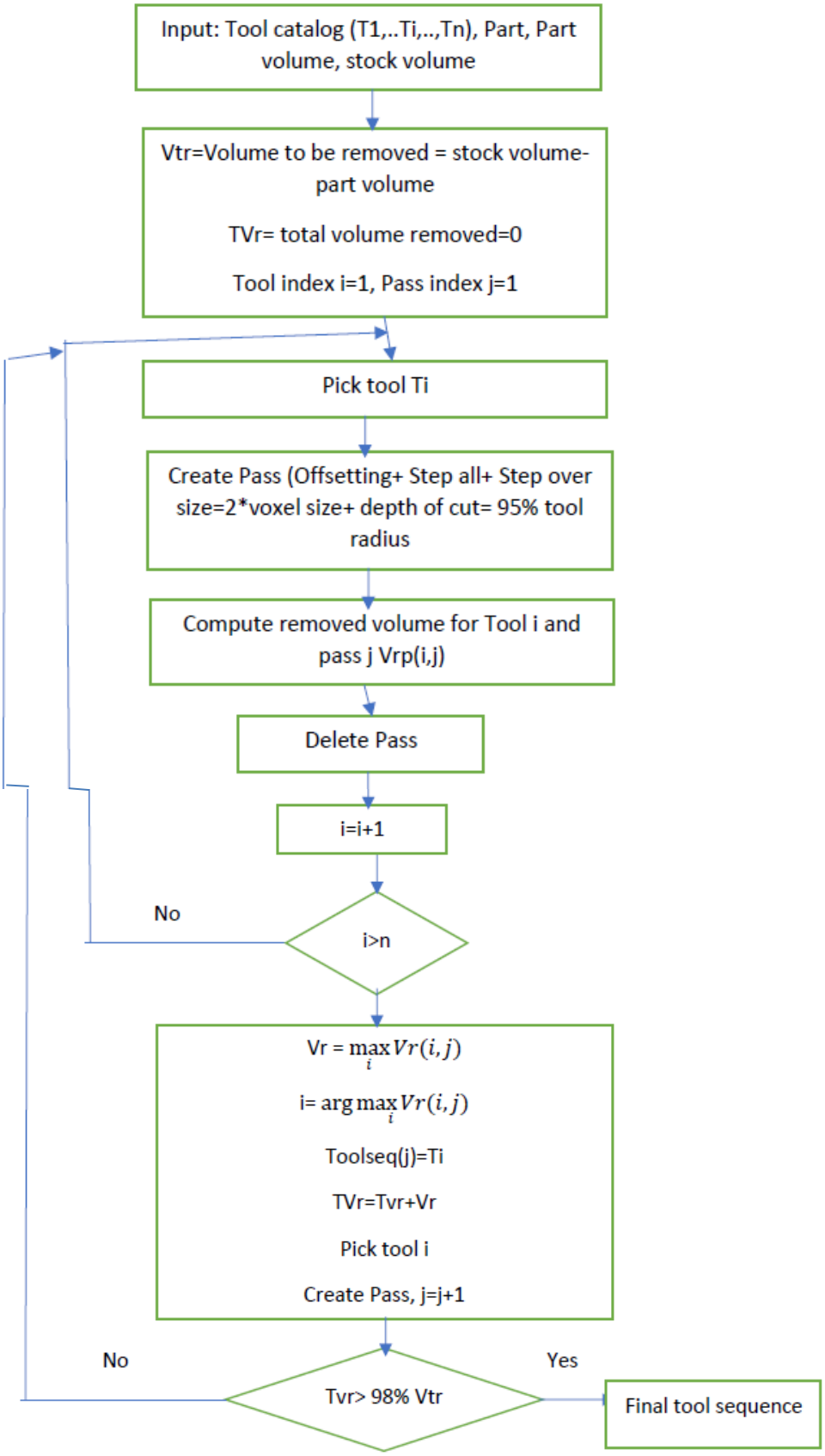


Figure 18: Algorithm's flowchart for volume-based optimization strategy

Error! Reference source not found. shows the main advantages and disadvantages of the volume-based optimization strategy. The method can be beneficial for an optimal solution that is easy to implement, has a relatively short run time and helps save memory space since it does not involve the generation of accessibility maps which requires generally larger memory allocations. However, the approach considers uniquely the material removal in each pass and excludes time aspects which are essential to assess productivity of the 5-axis machine and its embeddedness in overall equipment effectiveness.

Table 2: Advantages and disadvantages of volume-based optimization

Advantages	Disadvantages
- Simple implementation - All python commands are available - Computational effort: number of loops needed < N x K (N number of tools and K number of passes required to produce the part) - Potential of each tool is fully used - No need to go through all SP steps (Access maps, tool orientations, retractions …etc.) - Memory savings	- No consideration of **time aspects** - Tools might be **inaccessible** in some regions, since accessibility maps are not created yet - **Suboptimal** solution (local optimality) - Gives a general idea about the tools that can be used during machining but lacks accuracy in **path planning** - The tool path is generated automatically and there is **no way to influence it/ modify it,** since the goal is to have an algorithm that works independently from the part geometry

The inclusion of time aspects in the optimization strategy implicates the use of a different machining criterion that combines both volume and velocity parameters. A possible candidate for the aforementioned idea is the material removal rate.

4.2 Average Material Removal Rate (AMRR-based optimization strategy)

Material removal rate (MRR) represents the instantaneous amount of volume removed per time unit during a machining operation. From a traditional perspective, MRR is related to the axial depth of cut a_p, radial depth of cut a_r and federate v_f through Equation (3) and includes implicitly additional parameters such as spindle speed, number of teeth and feed per tooth.

$$MRR = a_p \cdot a_r \cdot v_f \tag{3}$$

The traditional approach is useful to analyze relatively simple toolpaths on plane surfaces where tool engagement has little variations from the mean values. However, 5-axis machining deals generally with freeform surfaces involving multiple tool reorientations due to the additional rotational joints offering more degrees of freedom and making the traditional path metrics inapplicable or obsolete. For this reason, voxel-based approaches can be implemented to transform the analytical methods into discrete GPU-friendly methods. Since the toolpath is composed of discrete points in the 3D space, the distance $d_{i|i+1}$ separating two consecutive points P_i and P_{i+1} can be computed as the norm of the difference of their corresponding position vectors:

$$d_{i|i+1} = \left\| \underline{P}_{i+1} - \underline{P}_i \right\| \tag{4}$$

To compute the total length of the tool path, the single distances are summed up over all N discrete points forming the path.

$$l = \sum_{i=0}^{N-1} \left\| \underline{P}_{i+1} - \underline{P}_i \right\| \tag{5}$$

The MRR can be derived from Equation (4) by considering the volume removal $V_{removed,i}$ after a single step and including the feed rate v_f.

$$MRR_i = \frac{V_{removed,i}}{d_{i|i+1}/v_f} \tag{6}$$

If the entire tool path is considered, the average MRR (AMRR) can be calculated using the total volume removal and the path length:

$$AMRR = \frac{V_{removed}}{l\ /v_f} \tag{7}$$

It is important to notice that AMRR is a measure of cutting efficiency that gives a general idea about how fast a cutter is removing stock volume per time, yet it does not necessarily represent accurately the different phases of volume removal and whether there are periods of time in which the volume removal stagnates or shows spike-like behavior. For instance, if the tool has a larger diameter than a surface concavity lying on its path, the MRR decreases drastically as the cutter cannot access (or has limited access to) the target region. An illustration of this case is depicted in Figure 19 where the removed volume is plotted against the machining time. The curve shows phases where the material removal

slows down or even remains constant and other phases showing a linear behavior of cumulative removed volume over time.

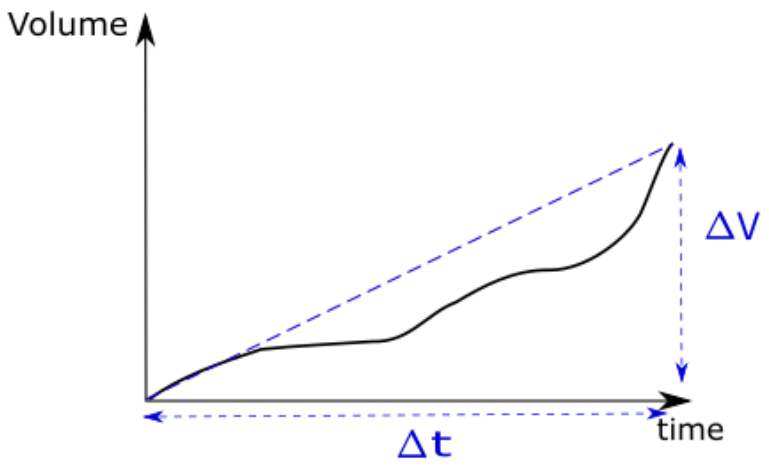


Figure 19: Example of a volume removal curve within a pass

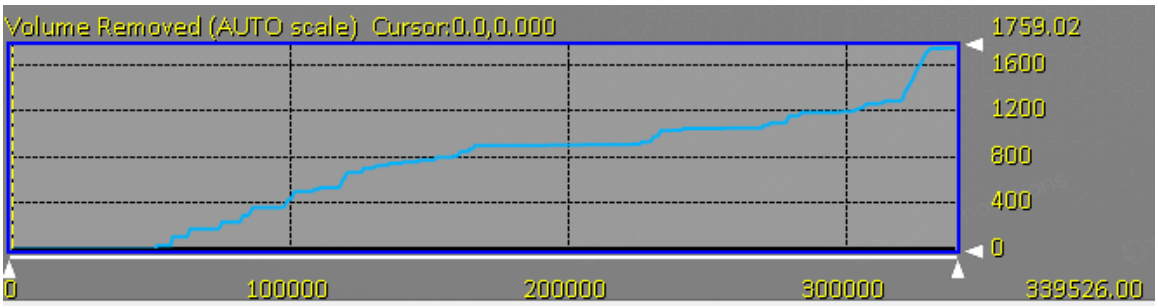
The AMRR can be adopted in the tool selection strategy as a single criterion to compare and quantify the efficiency of cutters in each tool pass. A similar greedy algorithm can be implemented to pick the tool that has the highest cutting efficiency -or in other words- is the fastest in doing its job. This algorithm should also overcome the disadvantages of the volume-based optimization strategy by including time aspects in form of volume velocity. Another advantage of this method is the ability to evaluate the time required for performing finishing passes and associated cost and quality tradeoffs. For the consistency of results, it is important for the selection criterion, in this case AMRR, to be representative for the removal rate over the entirety of the performed pass. Hence, MRR spikes or plateaus reduce the reliability of the average material removal rate as key figure for quantifying the efficiency of a given tool. In addition, with the available python capabilities of *SculptPrint*, it is computationally inconvenient to analyze separate sections

of a tool pass to derive a selection criterion. The evaluation is rather based on an analysis of the entire pass. Consequently, it is important to guarantee the uniformity of material removal rate by minimizing the time of ineffective cutting and eliminating the spike-like behavior in some pass sections.

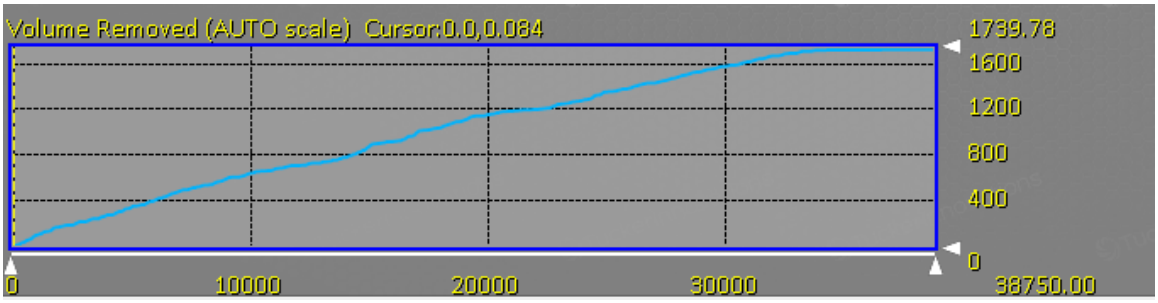
SculptPrint offers for these problems two solutions that can be implemented in two different stages of the typical process planning workflow.

4.2.1 Solutions for homogenizing the material removal

SculptPrint has a feature that allows setting a threshold for minimum material removal rate. 'Filter Path' is a command that eliminates the sections of the pass showing an MRR value below the defined threshold. By choosing adequate thresholds, stagnations in the volume curve can be reduced and the operation is more focused on the areas that can be machined more efficiently. Figure 20 exhibits a comparison between a) unfiltered path and b) filtered path after setting a threshold of 0.4 volume unit/step. The two curves show the cumulative volume removal at each step of the pass. The first plot includes several sections where volume removal is constant over a large number of steps, while the second plot shows an almost linear characteristic of volume removal since all irregularities have been eliminated through path filtering. Another benefit of this feature is the substantial reduction of the number of steps constituting the tool path (In the current example, the number of steps decreases from 339526 to 38750, which corresponds to 89% of reduction rate). Moreover, additional memory savings, the acceleration of accessibility maps generation due to data downsizing and a more accurate computation of effective cutting time are subsequent advantages of the used path filter.



a) Volume curve before filtering



b) Volume curve after filtering

Figure 20: Comparison of volume removal before and after filtering

According to the conventional workflow described in chapter 3, access maps generation and the creation of tool orientations are the steps that follow the path generation. Until this stage, the path is composed of connected time-independent steps. By creating the velocity profile, the software assigns to each step a timeframe Δt in such a way that a predefined MRR threshold is not exceeded. The algorithm starts with a given feed rate v_f and computes at each step the corresponding MRR using the formula given by Equation (6). If the computed MRR is larger than the preset threshold, a feed rate adjustment takes place, so that the resulting MRR remains below the threshold. The time assignment is executed with respect to the following scheme where $v_{f,adj}$ describes the adjusted feed rate:

$$\Delta t = \begin{cases} \dfrac{d_{i|i+1}}{v_f} & if\ MRR < Threshold \\ \dfrac{d_{i|i+1}}{v_{f,adj}} = \dfrac{V_{removed,i}}{Threshold} & if\ MRR > Threshold \end{cases} \quad (8)$$

The advantage of this feature is to limit the MRR spikes that can occur during a pass, which would contribute significantly to the uniformity of material removal.

4.2.2 Implementation of the AMRR-based strategy:

The AMRR-based strategy is an extension of the work done on the volume-based strategy and adopts the same idea of the greedy algorithm formulated in 4.1.1. Since the calculation of machining time is necessary to compute the AMRR, the implementation of this strategy must involve additional functionalities of *SculptPrint* including accessibility maps and tool orientations. The corresponding algorithm is implemented as follows:

1. Pick the largest tool in the library
2. Offset the part with the corresponding maximum depth of cut
3. Generate an automated toolpath with a given step over
4. Filter path
5. Generate connections: this step is necessary since the path filter breaks down the original path into steps having a higher MRR than a given threshold. The role of this functionality is to reconnect the remaining steps and form a single-line-path.
6. Generate accessibility maps: Check tool accessibility on each voxel and generate the (θ, β) angle map. Inaccessible points and sequences are subsequently removed

7. Create tool orientations: Choose a tool orientation strategy and generate a sequence of possible orientations to machine the part
8. Create velocity profile: assign time to each path step and plot volume curve
9. Get volume and time data and store in arrays
10. Linear curve fitting of volume/time plot: $V = A \cdot t + B$ and determine $R - squared$. In this case, A represents AMRR.

A, B and R^2 are determined as follows:

$$A = \frac{\sum_{i=1}^{n}(t_i - \bar{t})(V_i - \bar{V})}{\sum_{i=1}^{n}(t_i - \bar{t})^2} \quad (9)$$

$$B = \bar{V} - A\bar{t} \quad (10)$$

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(V_i - V)^2}{\sum_{i=1}^{n}(V_i - \bar{V})^2} \quad (11)$$

11. If $R^2 > 80\%$ then assign AMRR=A, otherwise AMRR=$\frac{\Delta V}{\Delta t}$
12. Pick the next largest tool and repeat the steps 2-11
13. After finishing all the tools in the library, pick the one which has the highest AMRR
14. Repeat 1-13 until the volume threshold is achieved

The flowchart depicted in Figure 21: Flowchart of AMRR-based strategy shows the algorithm structure of the AMRR-based strategy and contains one loop for trying out all tools and completing for each one the corresponding workflow within *SculptPrint*. A second loop is used to pick the tool having the highest AMRR and memorizing the

index of this tool, as it is the one to start with in the following loop. Advantages and disadvantages of the proposed method are summarized in Table 3.

Table 3: Main advantages and disadvantages of the proposed algorithm

Advantages	Disadvantages
- Machining time is implicitly included - Same concept as for volume based optimization - Not a lot of iteration loops - Enables the choice of the fastest tool in each pass	- Linearization of MRR may lead to information loss and is not always applicable e.g. in the case of non-uniform volume removal and random geometries - More computational effort than VBO since access maps, tool orientations and velocity profile have to be generated. - Values for MRR threshold are based on observations done through manual iterations - Higher memory usage

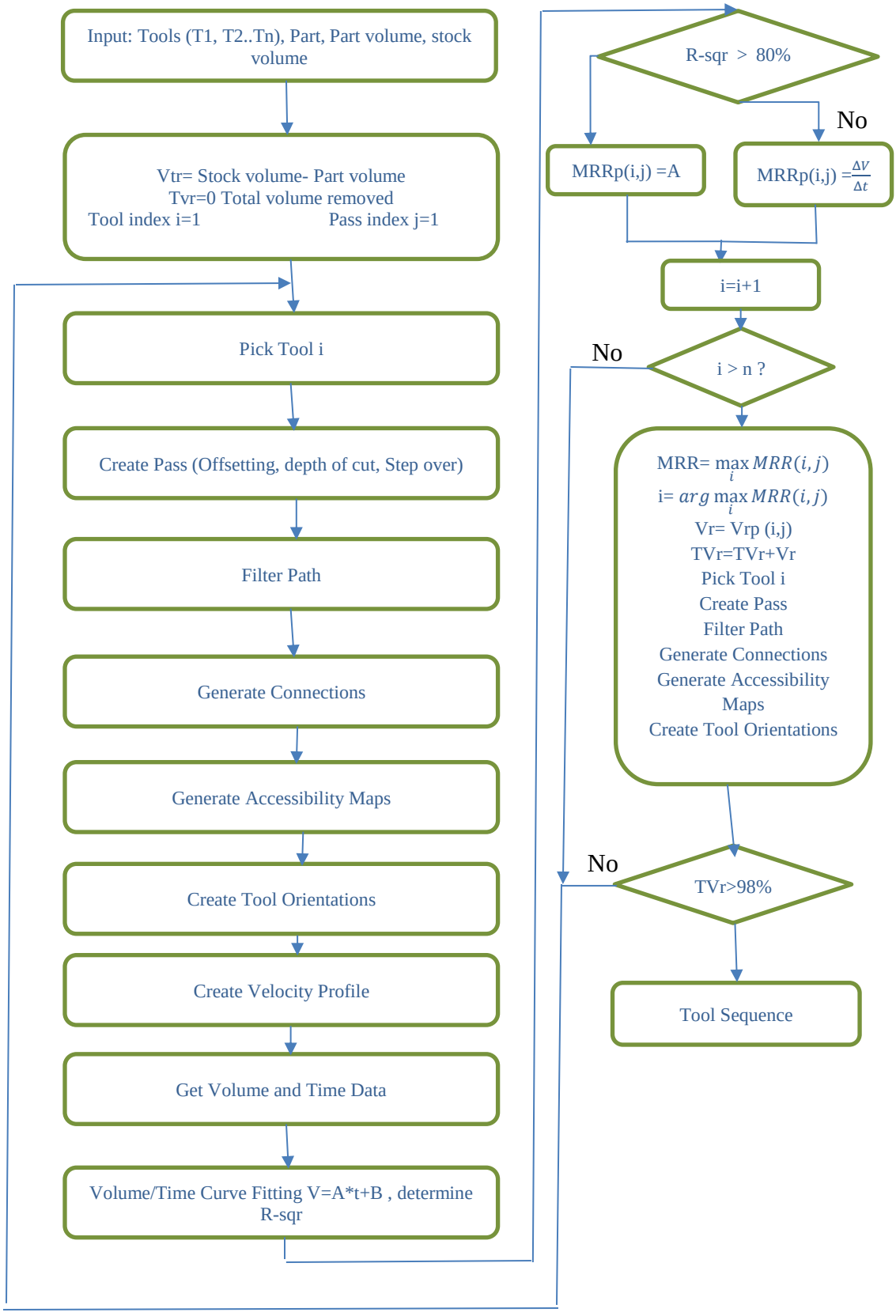


Figure 21: Flowchart of AMRR-based strategy

4.3 Two-Objective Optimization

A greedy algorithm is used for both volume and AMRR-based optimization. As mentioned in section 4.1.1, this technique leads to a global optimum only if the problem has an optimal substructure. That implies the algorithm does not investigate the possible paths in the decision tree and is satisfied with the choice maximizing the selection function at each stage of the optimization. Despite its simple implementation, this approach can lead to erroneous decisions having impact on the optimality of other factors such as machining time. Figure 22 shows a case where the algorithm has to select the optimal sequence to realize the passes N-1 and N. According to greedy paradigm, the sequence (T1-T1-T3) is optimal since the tool T1 has the highest volume removal (500 volume unit) when performing the pass N-1. However, if we consider the entire machining time required for realizing both passes N-1 and N, this combination is the slowest.

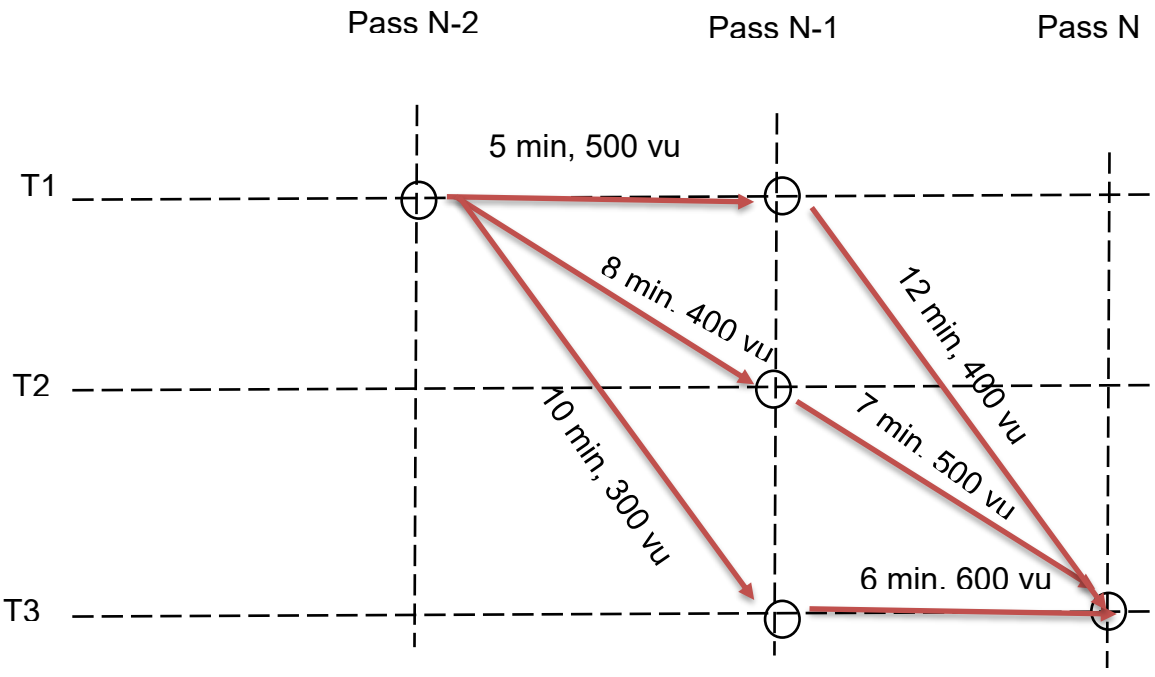


Figure 22: Machining time problematic with the greedy problem solving

To avoid the machining time problematic, a holistic approach involving all possible path selections should be considered. The problem can be treated a decision tree with a finite number of tools to select and passes to realize. At each pass, there is a possibility to use a tool that is as large as the tool used in the previous pass or smaller. This assumption ensures that the tool sequence starts with larger tools for roughing operations, whereas smaller tools are reserved for finishing. Since the optimization has to fulfill a percentage of completion after a certain number of passes, it is important to incorporate the volume removal resulting from each tool choice. Similarly, the time required to realize each pass should be considered and represents in each sequence combination a transition cost that is used to evaluate the overall machining time.

Figure 23 shows a graph-based approach to formulate the optimization problem.The vertices represent the possible tools to use in the jth pass. The edges are weighted transitions representing simultaneously the amount of volume ΔV removed by a tool T_i after using a tool $T_{i"}$ in the previous pass and the amount of time Δt required to perform the described operation. The graph has a directed acyclic characteristic: it consists of a finite number of vertices and edges, with each edge directed in such a way that it is impossible to start at a given vertex and loop back to it by following any of the possible sequences. The proposed multitree has a finite number K of stages where each stage describes a Pass with the index $j \in \{1,2 \dots, K\}$ and a finite number N of tools. Each tool occupies a vertex indexed by the tupel (i, j) with $i \in \{1,2 \dots, N\}$.

The optimal solution of this problem is the tool sequence with the minimum total machining time enabling a sufficient amount of volume removal. Based on this

formulation, the solution suggests a multi-objective optimization targeting both machining time and the percentage of completion.

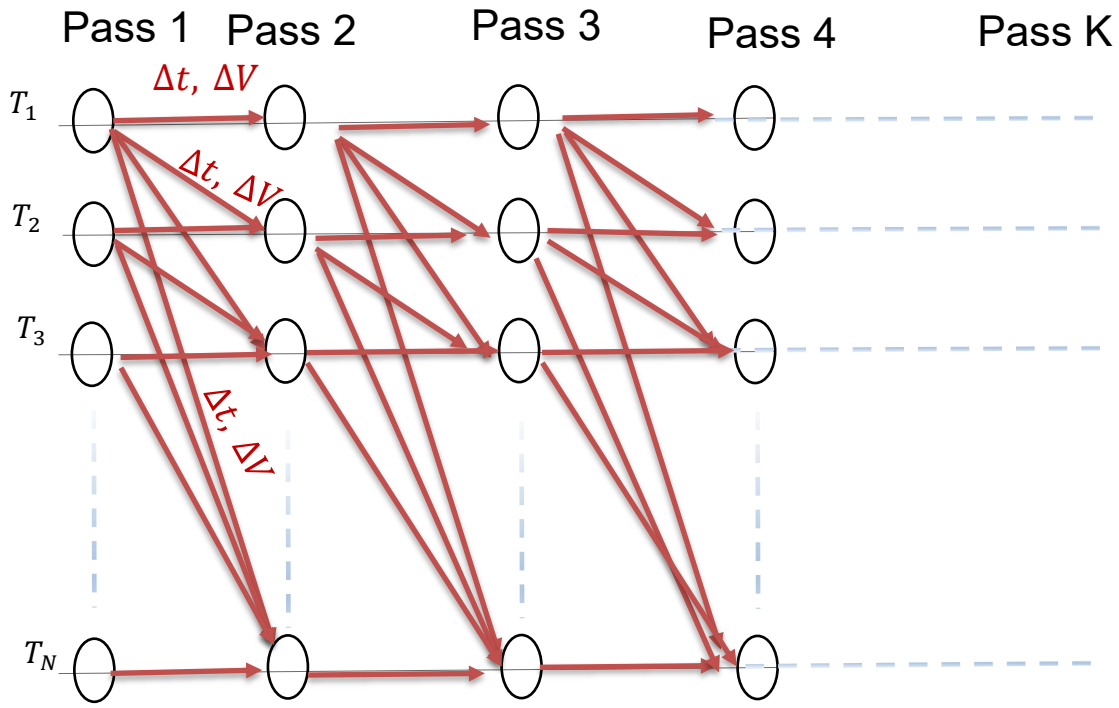


Figure 23: Multitree directed acyclic graph for machining time based optimization

A possible way to fulfill the optimization objectives is a vector formulation of a discrete optimal control problem (OCP). In control theory. OPC consists in controlling a state $\underline{x}$ with a control variable $\underline{u}$ to minimize a cost function J. In this case, $\underline{x_k}$ is the discrete state of the problem at the stage k and is defined as:

$$\underline{x_k} = [V_k] \tag{12}$$

where $V_k \in [0,100\%]$ expresses the cumulative percentage of volume removal. $\underline{u_k}$ represents the control variable and is defined as a 2D vector containing the current

percentage of volume removal ΔV_k and the used tool T_k. Equation (14) describes the dynamics of the system and ΔJ_k the transition costs expressed in machining time Δt_k. J is the cost function to optimize.

$$\underline{u_k} = \begin{bmatrix} \Delta V_k(\underline{x_k}, \underline{x_{k-1}}, u_{k,1}) \\ T_k \end{bmatrix} \tag{13}$$

$$\underline{x_{k+1}} = \underline{x_k} + [1 \quad 0]\underline{u_k} \tag{14}$$

$$\Delta J_k = \Delta t_k(\underline{x_k}, \underline{x_{k-1}}, u_{k,1}) \tag{15}$$

$$J = \sum_k \Delta J_k \rightarrow min \tag{16}$$

Since the current optimization has two objectives that are indirectly correlated, it is hard to find a single tool sequence that achieves simultaneously the highest volume removal and the shortest machining time. For this reason, it is important to prioritize one of the objectives and optimize the second one based on the constraints resulting from the fulfillment of the first optimization criterion. In other words, the algorithm should cluster the sequences that meet one of the objectives to a certain degree of tolerance and pick the best case, which optimizes the second criterion, out of the clustered sequences.

From a manufacturer's perspective, it is more reasonable to prioritize the volume removal criterion, as it reflects a percentage of completion of the manufactured part that has to be maximized. In opposition to that, prioritizing machining time can generate sequences leading to unfinished parts that still needs further processing through additional machining operations.

To generate feasible solutions, it is necessary to compute all possible paths in the graph and compute transition volume and time for the weighted edges. According to the conventional *SculptPrint* workflow, the computation of machining time requires the execution of multiple computationally intensive steps such as generation of access maps and creation of tool orientations. Additionally, the multi-tree structure implies the number of edges between vertices increases exponentially with the number of passes and tools. For simplicity's sake, the machining time is quantified as $\frac{l}{v_f}$ where l is the path length and v_f is the tool-specific constant feed rate. To ensure the machining efficiency, a path filter is applied prior to machining time computation.

Using combinatory techniques, the order of sequence generation can be arranged in such a way that the number of necessary passes decreases drastically and only non-redundant subsequences are removed between one combination and another as exhibited in Figure 24.

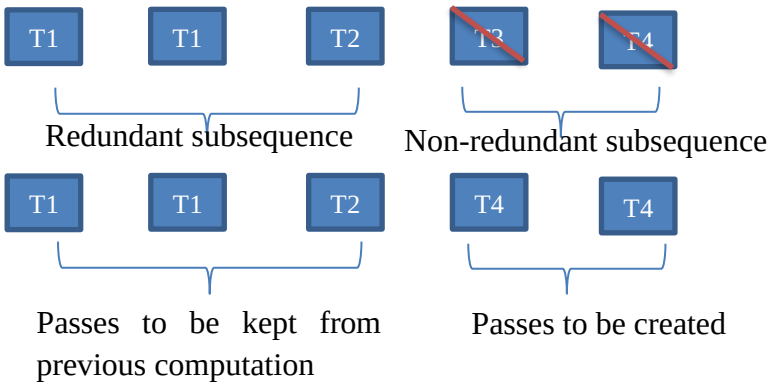


Figure 24: Removing non-redundant subsequences

The algorithm for the two-objective tool sequence optimization is formulated with the following outline:

1. For all possible sequences, do steps 2-4:
2. Create passes through automated tool path generation and with respect to the current tool sequence
3. Path filtering
4. Compute volume removal and machining time for each edge
5. Cluster the generated sequences based on volume removal
6. Pick the tool sequence with minimum machining time within the created cluster

The sequence clustering is realized by selecting the combinations leading to a total volume removal higher than a given threshold, or by ranking the top 5 sequences or by finding the sequence having the highest volume removal and searching the other sequences located within a tolerance window. An illustration of sequence clustering is given by Figure 25 where the optimal solution is selected based on machining time minimization.

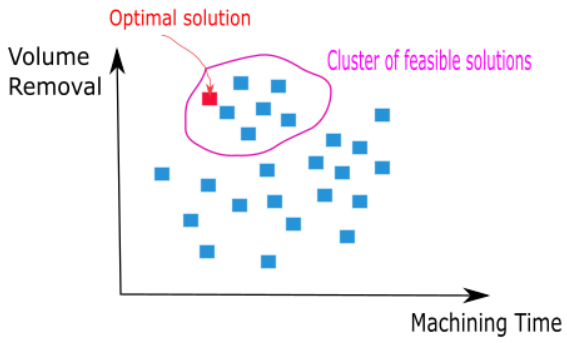


Figure 25: Clustering of feasible solutions and selection of the optimal solution

Special Case

Suppose the DAG multitree depicted in Figure 23 starts with a single possible choice and similarly, all edges in the final pass converge to a single choice. For instance, the

machinist is certain of using the largest tool during the first pass and the last pass has to be realized with a particular cutter for surface finish purposes. In this case, the problem is transformed into a single-source single-sink directed acyclic graph. Moreover, each vertex in the graph has the possibility to move to a vertex from the same level or lower. The literature review shows that the current problem structure is very similar to a known graph-based problem referred to as Manhattan Tourist Problem (MTP) [68].

MTP seeks a path from source to sink and can move only eastward and southward with the goal of seeing most number of attractions in the Manhattan grid.

Figure 26 shows the Manhattan grid with red stars symbolizing the presence of touristic attractions. An additional weight is assigned to each edge (street section) hosting an attraction and the weight value is dependent on the importance of this attraction. Consequently, solving MTP consists in finding the longest path (or the path with the largest weight) from source to sink.

Dynamic programming is a powerful tool for solving MTP problems. It breaks down the entire problem into identifiable sets of subproblems and addresses them one after the other beginning from the smallest. The algorithm uses the solution of smaller problems to tackle larger ones based on the dependencies of subproblems. Analogously, the same algorithm can be implemented in the special single-source single-sink tool sequence optimization problem. The unique difference between MTP and the tooling problem described in this section is the possibility to move along the diagonals, since the decision tree has branches connecting large tools not only to the next largest but also to the smallest.

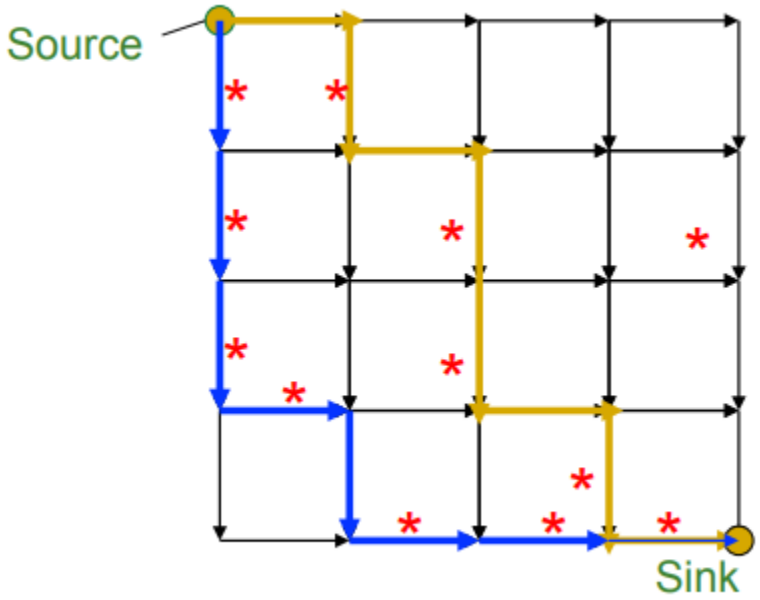


Figure 26: Manhattan Tourist Problem

The implementation of dynamic programming can be beneficial for finding the tool sequence with the highest volume removal. The recursive algorithm is formulated as follows:

Longestpath (N=number of tools, K=number of passes)

$A_1 \leftarrow \boldsymbol{Longestpath}(1, K-1) + \ edge\ length\ from\ (1, K-1)\ to\ (N, K)$

$A_2 \leftarrow \boldsymbol{Longestpath}(2, K-1) + \ edge\ length\ from\ (2, K-1)\ to\ (N, K)$

$\vdots$

$A_N \leftarrow \boldsymbol{Longestpath}(N, K-1) + \ edge\ length\ from\ (N, K-1)\ to\ (N, K)$

return max $(A_1, A_2, \ldots A_N)$

Algorithm 2: Dynammic Programming algorithm for finding the longest path in the tooling problem

Table 4: Advantages and Disadvanteges of the two-objective optimization strategy

Advantages	Disadvantages
- Holistic approach - Global optimality - Volume and time aspects are explicitly considered - Ability to make a trade-off in the final stages if the remaining passes would take long time	- All costs should be calculated first - Number of stages is unpredictable and depends upon stopping condition and part geometry - Computationally (very) intensive - Multiple constraints must be considered to increase the efficiency of the algorithm - Tool path planning not influenceable

Table 4 highlights the main advantages and disadvantages of the multi-objective method with focus on global optimality as main benefit and computational complexity as major drawback.

4.4 Experimental Setup

One of the goals of this study is to obtain quantitative data that demonstrate the performance of the formulated optimization algorithms on local and virtual platforms. The performance assessment serves as benchmarking for computation time required to execute the several kernels of each algorithm. Due to the voxel-based configuration adopted by the software tools used in this thesis, most of the kernels are highly parallelizable, hence the benchmarking study compares the performance of several GPUs. To this end, 3 GPU

platforms are considered: one local and two virtualized. One of the virtualized platforms is hosted by GT-Virtual Lab and the second is a cloud computing solution offered by Amazon Web Services in form of Infrastructure as a Service (IaaS).

4.4.1 Georgia Tech Virtual Laboratory

Georgia Tech Virtual Laboratory (Vlab) is a computing infrastructure that provides Georgia Tech students, faculty and staff remote access to virtual machines hosted by the school from anywhere in the world with any computer having internet access. Vlab offers a variety of software applications and physical resources that can be used 24 hours/7days. The first introduction of GPU-accelerated servers took place in 2012 [59] to enable students to run graphics applications using a GPU pass-through via shared desktops on the internet and conduct research activities without individual software licence requirements. For the implementation of GPGPU pipeline, powerful hardware with sufficiently many CUDA cores is required. A large number of parallel processing units allows limiting performance losses caused by an increasing number of users or layers of abstraction. To this End, the GPU deployed for this study is nVIDIA ® Tesla ® M60 which is a high-performance GPU-accelerator recommended for the implementation in virtualized workstations.

Figure 27: Tesla M60 graphic card

Figure 28: Georgia Tech Virtual Lab

Figure 27: Tesla M60 graphic card and Figure 28: Georgia Tech Virtual Lab show respectively photos of Tesla M60 card deployed in this work and the data center of GT virtual lab.

As mentioned in 2.6, the virtualization of hardware is ensured by a hypervisor that manages the resource sharing and the accessibility modes. The hypervisor chosen for this thesis is Citrix XenServer 6.5 SP1 and is installed on Windows 2012 R2. The advantage of the chosen hypervisor is that a direct access to GPU is granted without additional layers of abstraction causing potential system slowdowns. SculptPrint is also installed on the virtual machine to compare performance with local platforms. The implementation hierarchy is given by the virtualization scheme in Figure 29.

For absolute performance, this configuration may not be the optimal choice since multiple user sessions can run simultaneously, and therefore the performance is confined to the number of CUDA units granted to each user.

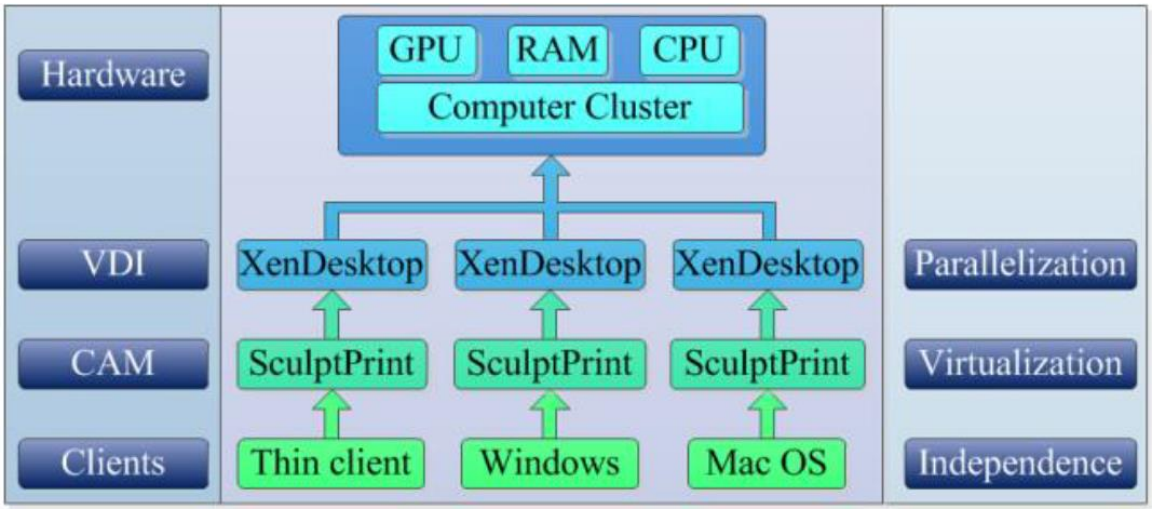


Figure 29: Virtualization hierarchy of SculptPrint [59]

4.4.2 Amazon Web Services

Amazon Web Service (AWS) is a cloud services platform, offering computing power, data storage and other solutions, such as management tools, artificial intelligence packages, business intelligence and software development platforms. This IT infrastructure delivers services as an on-demand utility and pay-as-you-go pricing [69].

The AWS product relevant for this work is the Elastic Compute Cloud EC2. It presents a virtual computing environment with the ability to launch instances through web-based Amazon Machine Images (AMI). EC2 has different instance types depending on the desired compute power and use case. For graphics application, AWS offers G2 instances and P2 instances. G2 instances provide both graphics and compute allowing the user to run both SculptPrint python scripts and SculptPrint proper. P2 instances provide only compute

power. Only SculptPrint python scripts can be run on these instances. The instance used in this work is g2.2xlarge and has an nVIDIA GRID K520 graphic card. The instances are launched on remote desktops and use hardware placed on different geographical locations, conventionally stored in large data centres as depicted in Figure 30.

Figure 30: AWS data center in India

4.4.3 Hardware Comparison

The specifications of the deployed graphic cards are summarized in Table 5, while Table 6 contains the main non-graphic specifications of the considered computing platforms.

Table 5: Graphic cards specifications

Specifications	Local Desktop	GT Virtual Machine	AWS G2 Instance
nVIDIA Product	Quadro M4000	Tesla M60	GRID K520
Number of CUDA cores/ GPU	1664	2048	1536
Total CUDA Cores	1664	4096	3072
Memory Size(GB)	8 GDDR5	16 GDDR5	8 GDDR5
Memory Bandwidth (GB/s)	192	320	380

Table 6: System specifications for the considered computing platforms

Specifications	Local Desktop	GT Virtual Machine	AWS G2 Instance
Processor	Intel® Xeon® Processor E52623 (3 GHz)	Intel® Xeon® Processor E52680 (2.7 GHz)	Intel® Xeon® Processor E52670 (2.6 GHz)
RAM Size	16 GB	16 GB	16 GB
Hard Drive Size	1 TB	60 GB (SSD)	60 GB
Operating System	Windows 2016 R2	Windows 2012 R2	Windows 2012 R2

CHAPTER 5. RESULTS

In this chapter, the results of the different optimization algorithms are presented. The focus lies not only on the generated optimal tool sequences but also on the benchmarking of computation time on different platforms and their associated GPU performance. The influence of the variation of input parameters on the tool selection process is also investigated. Changed parameter are step over of the generated tool path (path resolution) and the voxel size of the considered part (mesh resolution).

The deployed algorithms, with their corresponding optimization criteria, are illustrated in Figure 31. For each optimization objective, sample parts are chosen to run simulation on, and generate a tool sequence that is capable of transforming the material stock into the target part.

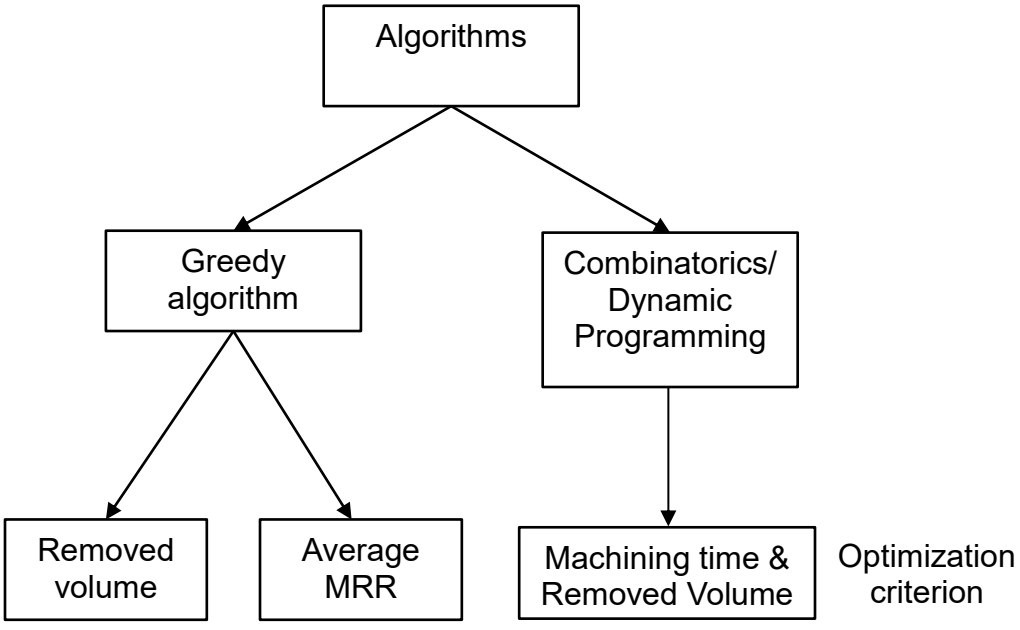


Figure 31: Summary of tooling optimization algorithms

To begin with, 3 sample parts are selected to run the volume-based greedy algorithm on. The part selection is based on geometric complexity and aims at proving not only the reliability of *SculptPrint* on dealing with complicated freeform surfaces, but also the concordance of the generated results with traditional approaches.

Table 7 gives an overview of the used parts for the simulation of the first optimization algorithm. A triangular mesh in STL format and a voxel model for each part are illustrated. The parts are sorted in descending order of geometric complexity. For instance, the candle holder shows a large number of concave surfaces with limited tool accessibility and an empty inside volume. The surfaces in the tea pot are more convex, which reduces the probability of local gouging, even with relatively large tools. The third part presents an inner pocket with rounded corners whose tool sequence can be intuitively obtained through traditional methods. The candle holder is also the part used for both AMRR-based strategy and the two-objective optimization.

The tools used for this study are ball-end mills with cylindrical tool holders. They can be extracted from *SculptPrint* default library for metric tools. The maximum axial depth of cut of each tool is assumed to be 95% of the corresponding tool radius. The feed rates are chosen from the machining book here [70]. Table 8 shows the tool list considered in this work, where each tool is referenced by an ID-number (T1→ T6).

\

Table 7: System specifications for the considered computing platforms

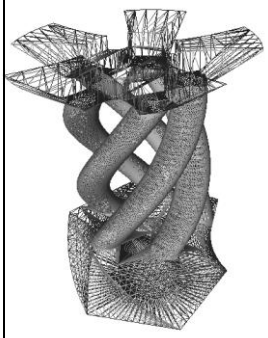
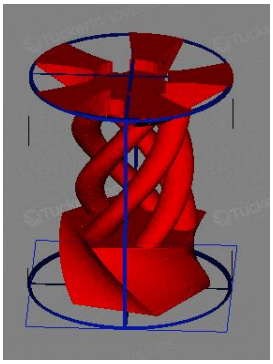
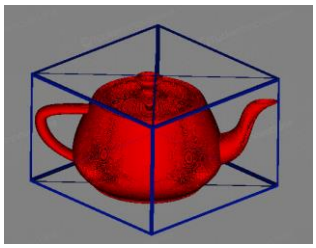
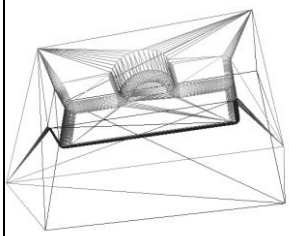
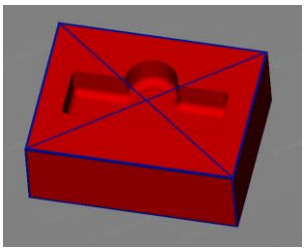
Part	Dimensions XYZ (mm)	Volume (mm^3)	STL model	Voxel model
Candle holder	48 x 49 x 58	21,275		
Tea pot	46 x 46 x 31	25,619		
Pocket	102 x 76 x 38	259,625		

Table 8: Example of a tool catalog

Tool ID	Tool diameter	Axial depth of cut (mm)	Feed rate (mm/min)	Effective cutting diameter (90-degree orientation) (mm)
T1	0.5 in → 12.7 mm	6.03	558.8	12.68
T2	5/16 in → 7.93 mm	3.77	495.3	7.92
T3	1/4 in → 6.35 mm	3.01	457.2	6.34
T4	3/16 in → 4.76 mm	2.26	431.8	4.75
T5	1/8 in → 3.17 mm	1.50	381	3.17
T6	3/32 in → 2.38 mm	1.13	355.6	2.37

5.1 Results of Volume-Based Optimization Strategy

The candle holder is imported in *SculptPrint* as STL file, and voxelized with a 0.5 mm side length of the cubic voxel. After importing the tool catalog, tools and voxelized part are fed to the volume-based optimization algorithm, described in Figure 18, as input variables.

In each iteration loop, one tool is selected from the library. Next, an offsetting of the part geometry with respect to the maximum depth of cut of each tool is executed. A graphical illustration of the offset volume created through several tools is shown in Figure 32. Next, the tool path generation is realized with a step over corresponding to 50% of each tool radius. Eventually, the algorithm picks the tool having the highest volume removal after finishing the pass. The process runs iteratively until 98% of the removable volume is achieved.

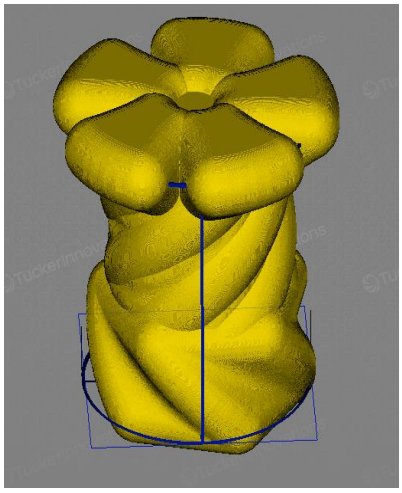
Figure 33 and Figure 34 show the volume removed by the tools (T1, T2…T6) during each pass. Based on the greedy paradigm, the tool with the highest volume removal is selected. The optimal tool for a given pass is highlighted in green and is the one to start with in the subsequent iteration. This implies, if the tool sequence shifts to a smaller tool within an iteration loop, larger tools are kept out of consideration in the decision-making process for future iterations.

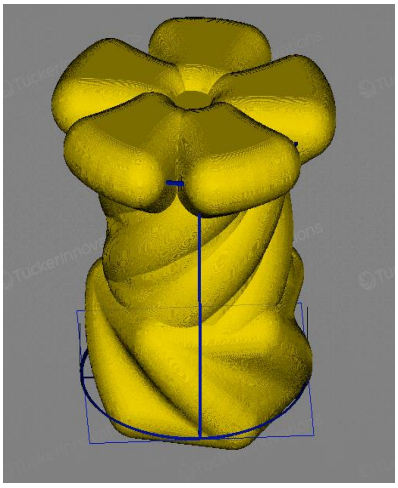
For the example of the candle holder, 10 passes were necessary to achieve 98% of the entire removable volume. The optimal tool sequence is given by the following table:

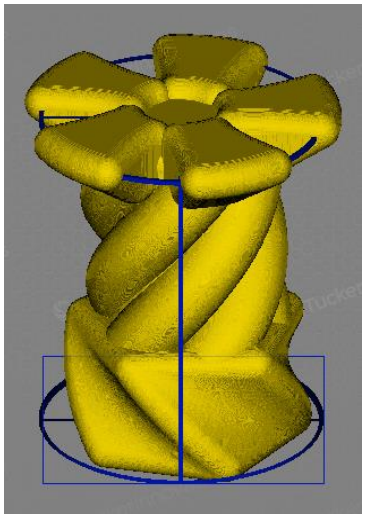
Table 9: Optimal tool sequence for the candle holder example

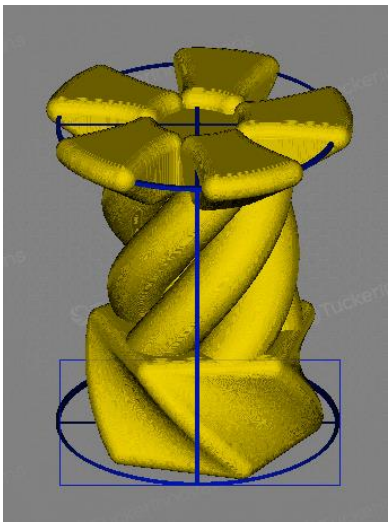
Pass	1	2	3	4	5	6	7	8	9	10
Tool	T1	T1	T4	T4	T5	T5	T5	T5	T6	T6

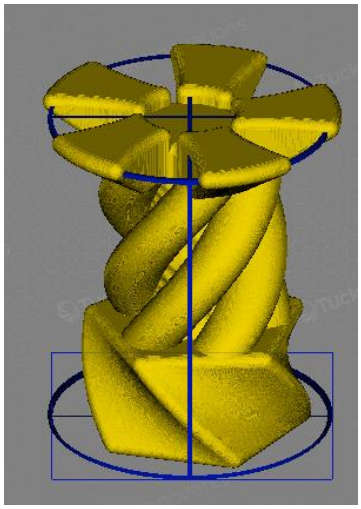
As shown in Table 9, T2 and T3 are excluded from the tool sequence, since they had lower volume removal in the 3rd pass than T4. The potential of T4 is exhausted in the 5th pass and a shift to T5 takes places. This shift is caused by the decrease of accessibility range of T5 with the appearance of concavities having relatively smaller curvature radii in the stock material, as demonstrated in Figure 37. The creation of the inside empty volume is realized by tool T6, the smallest tool in the library.


a)


b)


c)


d)


e)

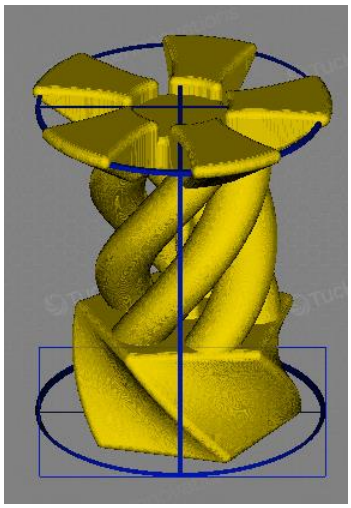

f)

Figure 32: Offsetting realized by tools T1 a), T2 b), T3 c), T4 d), T5 e), and T6 f)

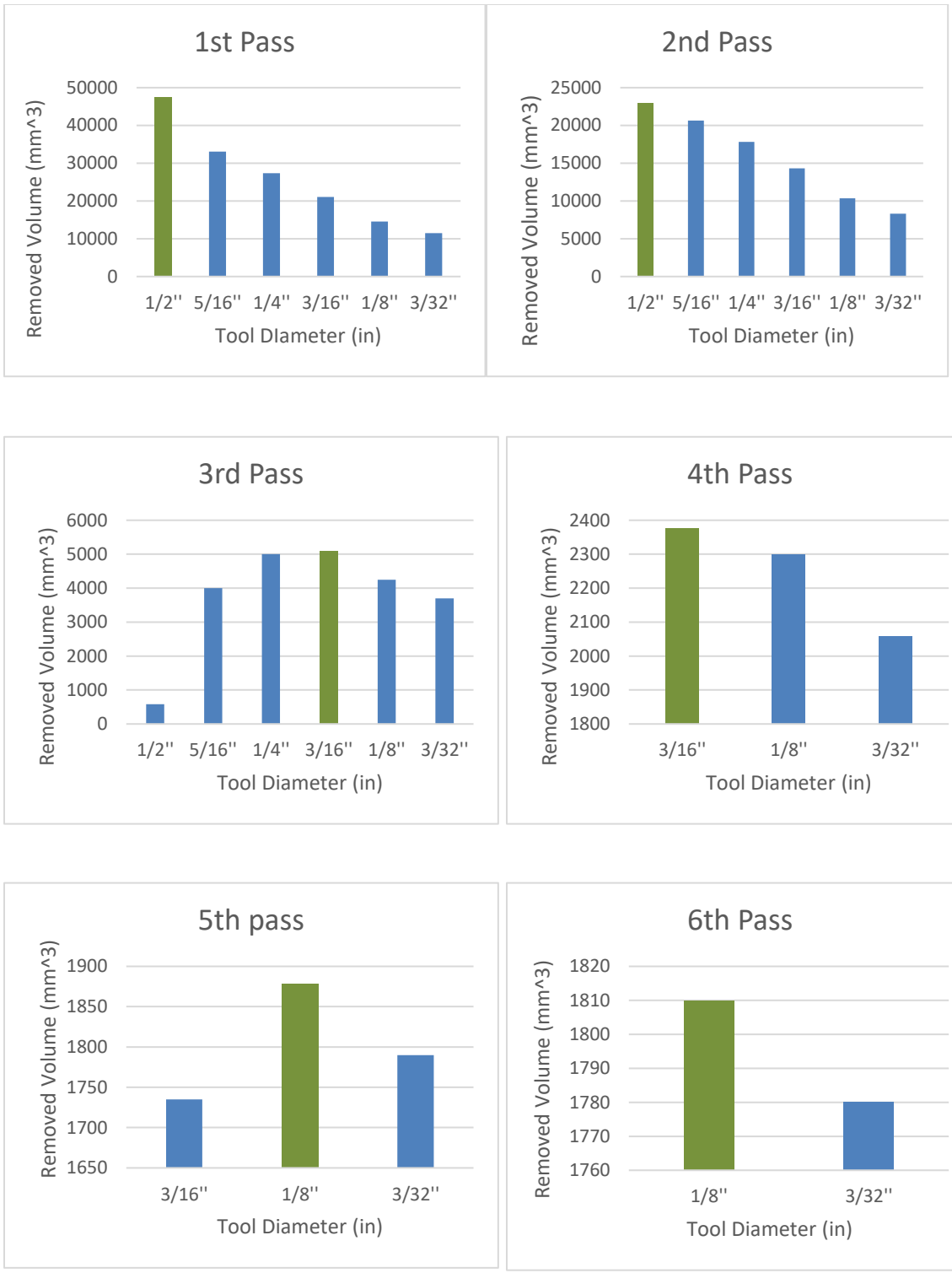


Figure 33: Removed volume within the first 6 passes and the selection of the optimal tool

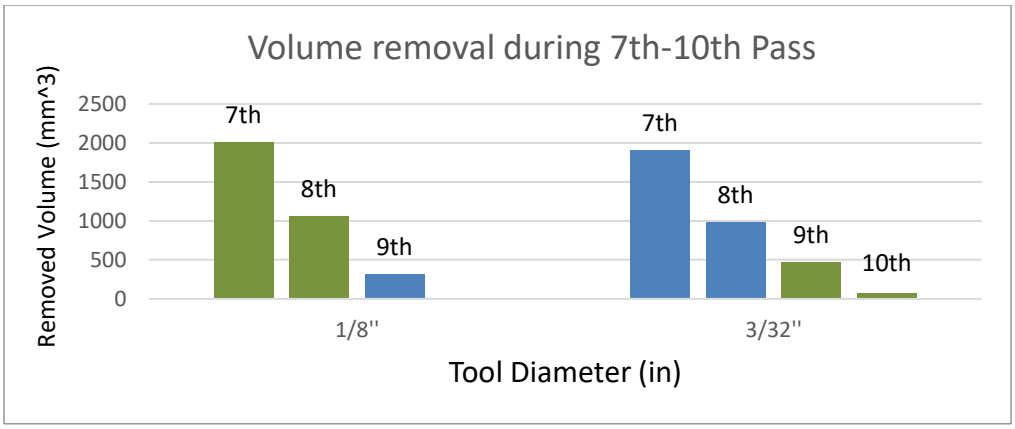


Figure 34: Removed volume within the first 6 passes and the selection of the optimal tool

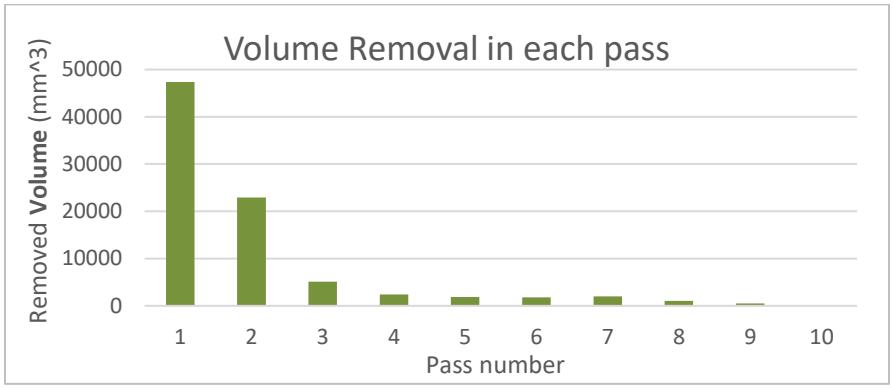


Figure 35: Volume removal through the optimal tool sequence

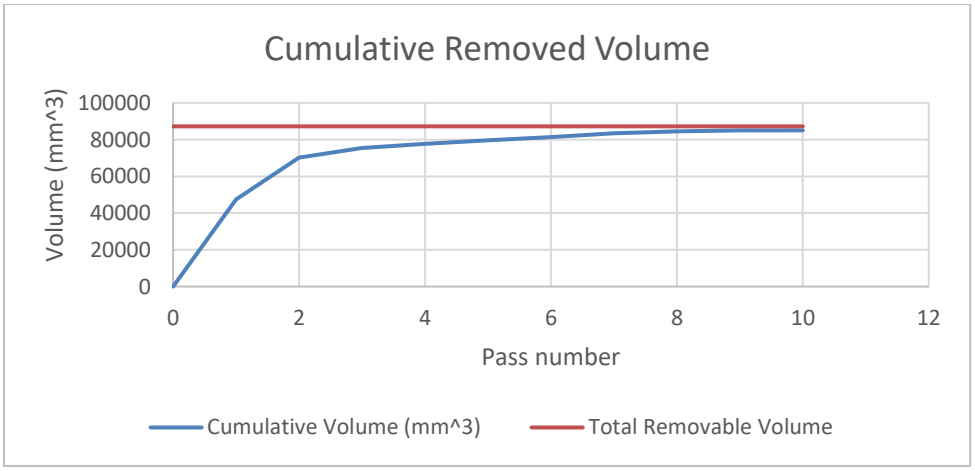


Figure 36: Cumulative volume removal for the candle holder

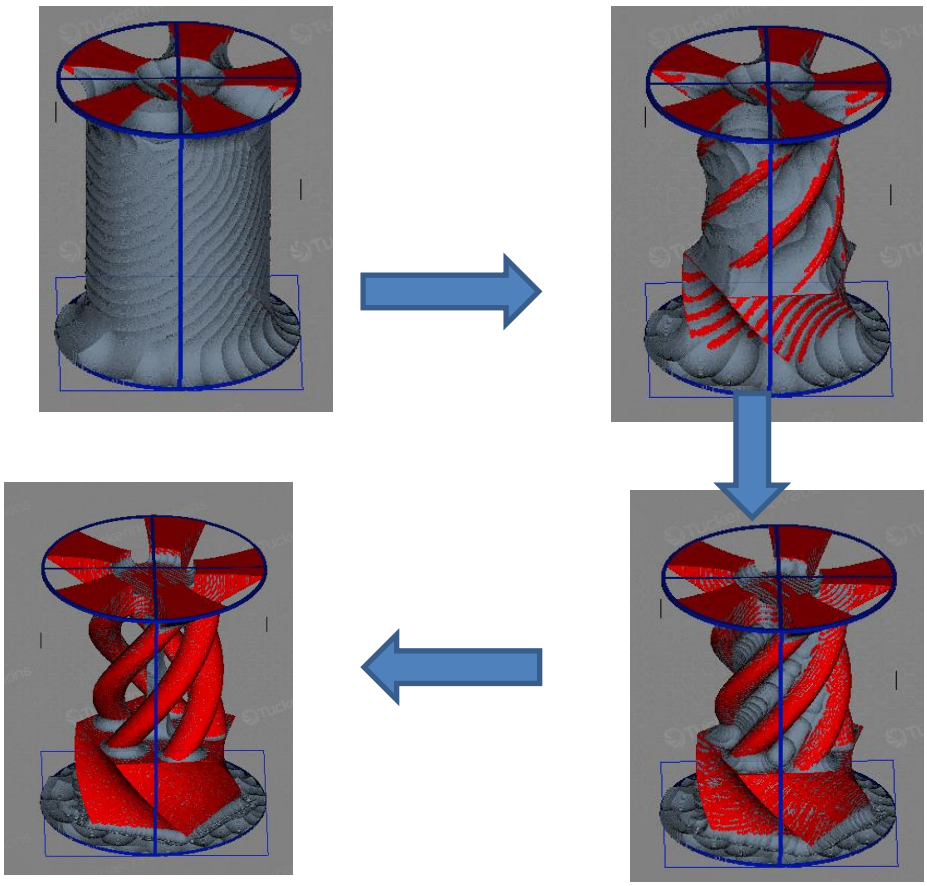


Figure 37: Shape transofrmation during diffrent stages of machining

Figure 36 shows the cumulative volume removal for the candle holder. The curve is obtained through discrete integration of the values of volume removal during each pass, exhibited in Figure 35. The plotted volume curve has an asymptotic characteristic, as it converges to the limit set by the total removable volume. The curve behavior is similar to a low-pass filter step response, where the growth rate is maximal at the beginning and decreases continuously with time, until stabilizing at the end. For the current example, this corresponds to the roughing and finishing phases, present in conventional machining operations.

As following candidate, the teapot model is selected, and the greedy algorithm is run with the same machining and rendering parameters as for the example of candle holder. The optimal tool sequence generated for the tea pot is given by the following table:

Table 10: Optimal tool sequence for the tea pot example

Pass	1	2	3	4	5	6
Tool	T1	T3	T4	T5	T6	T6

With only 6 passes, the teapot can be machined from the original stock material, where T1 and T3 perform the roughing operation, while T4, T5 and T6 the finishing passes.

The same algorithm is executed on the third part which presents a 2.5D pocket with rounded corners. The dimensions of the pocket are chosen in such a way that the tool sequence can be intuitively determined. For instance, the pocket has a depth of 0.65in=0.25in+0.25in+0.15in, which corresponds to the sum of twice the radius of the T1 tool and once the radius of T2 tool. The choice of these values suggests that the two first passes are realized by T1, and the third by T2, so that the pocket depth is completely achieved. Additionally, the bottom of the pocket has rounded edges with respect to the vertical faces. The radius of the corresponding curvature is 0.16in, which enables T2 to be engaged with full cutting depth (radius of the ball-end mill). The upper corners of the pocket have a radius of 0.13in and the lower a radius of 0.6 in, corresponding respectively to those of T4 and T5. A plausible tool sequence for the current example is T1→T1→T2→T4→T5.

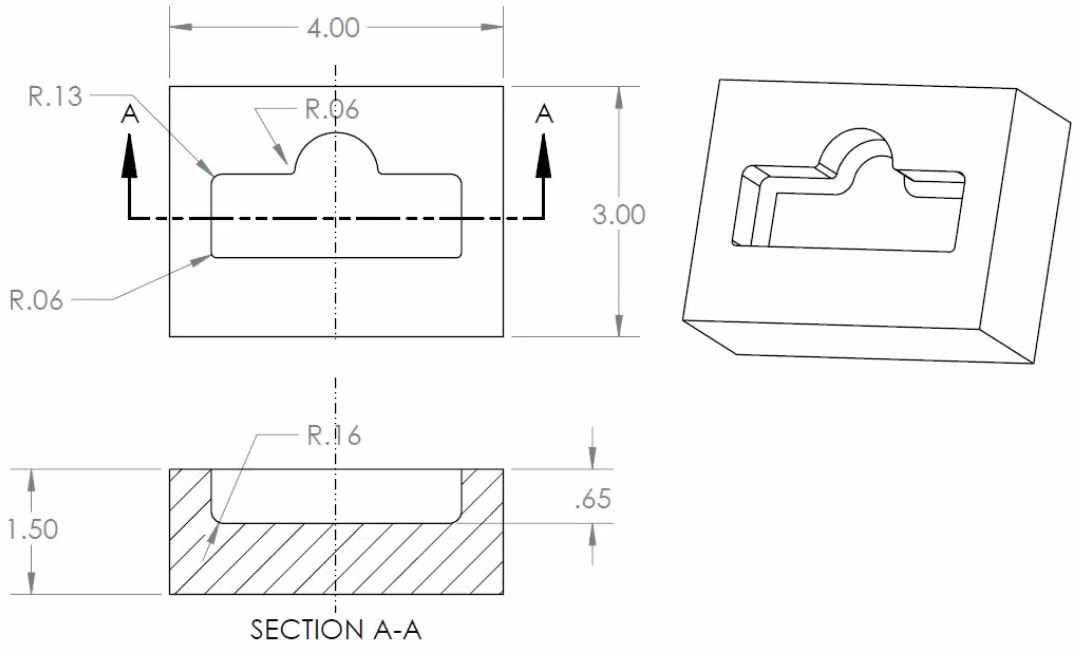


Figure 38: Dimensions of the 3rd part

The generated tool sequence by the greedy algorithm is introduced in the following table:

Table 11: Optimal tool sequence for the pocket example

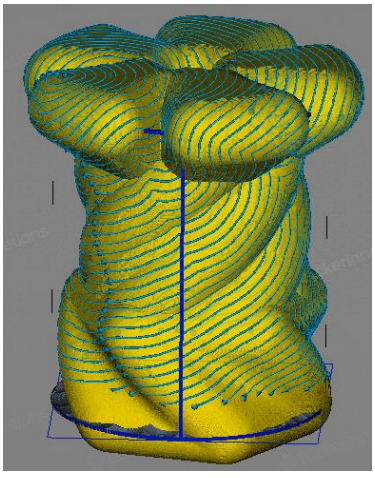
Pass	1	2	3	4	5
Tool	T1	T1	T2	T6	T6

The optimization results are compatible with the trivial solution introduced in the previous paragraph. The initial 3 passes are realized respectively by T1, T1 and T3, which is similar to the intuitively obtained tool sequence. However, the two last passes are not realized by T4 and T5, on the contrary, by T6. This is explained by the nature of the automated tool path generation functionality. *SculptPrint* generates a path that covers the entire offset volume with circular patterns starting from the point with the highest Z-coordinate. If all

rounded corners are affected by the path, T4 is inaccessible in the lower corners, and has therefore lower volume removal than smaller tools. One other explanation is the voxel-based approximation of the parametric surfaces, which limits the accessibility of tool T5 and makes of T6 the optimal choice.

5.1.1 Variation of Tool Sequence with Path resolution

In this section, the dependency of tool sequence selection upon the path resolution is investigated. The objective is to determine the impact of a changed step over on the generated solutions. To this end, the optimization algorithm is simulated on the candle holder example, but this time with 5 different step over values. The chosen step over is assumed to have a linear relationship ($stepover = \alpha \cdot r_{T_i}$) with the currently used tool radius r_{T_i}, where $\alpha \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$. The influence of the changed step over on the path resolution is exhibited in Figure 39. With higher resolution, the software has to produce more circular patterns to cover the totality of the offset volume, and the tool marks are more densely overwritten by adjacent paths.



(a)

(b)

Figure 39: Path resolution using T1 with (a) $\alpha = 0.3$ and (b) $\alpha = 0.7$

Table 12: Variation of optimal tool sequence with changed step over

α ↓ / Pass index →	1	2	3	4	5	6	7	8	9	10
0.1	T1	T1	T4	T4	T5	T5	T5	T5	T6	T6
0.3	T1	T1	T4	T4	T5	T5	T5	T5	T6	T6
0.5	T1	T1	T4	T4	T5	T5	T5	T5	T6	T6
0.7	T1	T1	T4	T5	T5	T5	T5	T5	T6	T6
0.9	T1	T1	T4	T5	T5	T5	T5	T5	T6	T6

Table 12 shows the impact of varied path resolution on the generated solutions. For $\alpha \in \{0.1, 0.3\}$, the optimal tool sequence is identical to the one generated with $\alpha = 0.5$ in the previous section. Nevertheless, solutions with larger step over values ($\alpha \in \{0.7, 0.9\}$) suggest an earlier shift from T4 to T5 during the 4^{th} pass. The change in the optimal sequence is due to the path geometry and its impact on the swept voxels in each pass. In fact, with increasing inter-path distance, the tool skips many more voxels and the achievable volume per pass decreases consequently. Smaller tools with higher path resolution are, therefore, selected by the optimization strategy.

With higher path resolution, the system has to reserve more memory on the hard disk to store *SculptPrint* files, as shown in Figure 40. A further implication of this factor is the computation time required to run the optimization algorithm and generate the tooling results. To demonstrate the relationship between computation time and path resolution, the duration taken by the algorithm to output the optimal tool sequence with each value of α is measured. For benchmarking purposes, the same operation is repeated on different

computing platforms. The performance of different GPUs is compared to gauge the profitability of the use of each platform and its correlation with hardware specifications.

Figure 40: Relationship between SP file size and path resolution

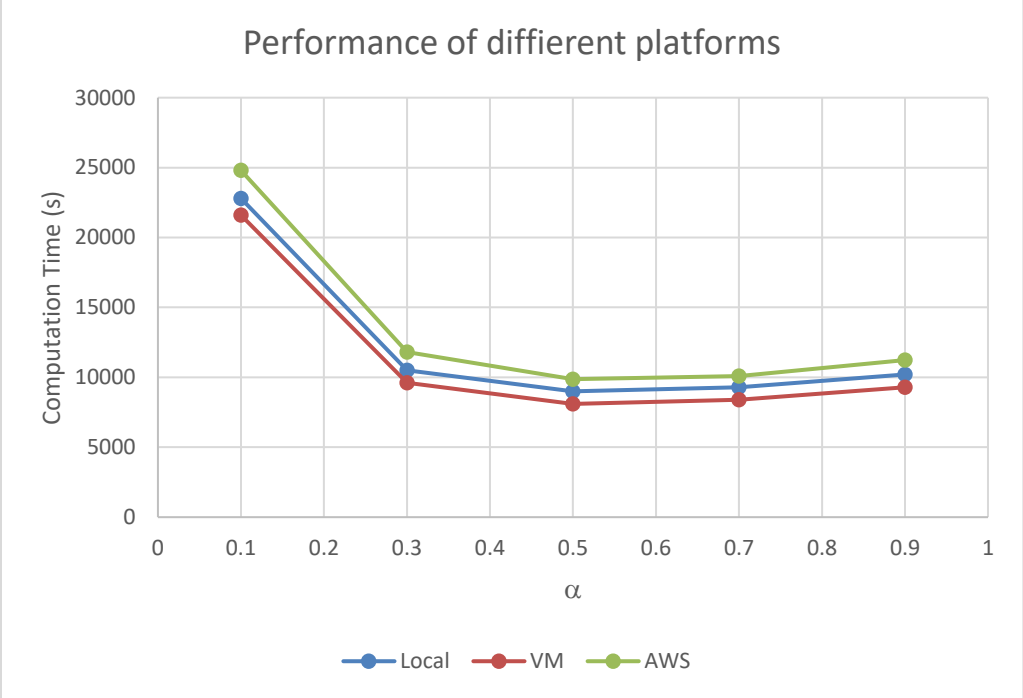


Figure 41: Benchmarking results

Figure 41 shows the correlation between computation time and the chosen path resolution for different platforms. The 3 plotted curves exhibit the same behavior with a particular performance offset. For the lowest value of α =0.1, the system requires the longest

computation time to converge to the optimal solution. In this case, the software has to perform a large number of steps and accordingly multiple sweeping and consolidation operations for each processed voxel on the path. The path update and creation of tool marks is also a time-consuming process associated with GPU overhead [19] . The curves show a local performance optimum at $\alpha = 0.5$. For higher values of α, the performance is slightly slower than the optimum, although the path resolution is lower. To understand the source of this difference, the main computing task can be decomposed and the time taken by each instruction can be investigated. In each pass, the software offsets the part with the given tool radius, then generates an automated tool path. The generation of tool path is composed of two operations: elements sweeping and end volume generation. Since the offsetting is a path-independent operation and is consequently not related to α, the sweeping and volume generation time are measured for different values of α. It is also important to notice that the tool T5 is the most frequently used tool in the generated sequences. Therefore, the computation time for path generation with T5 is investigated. The used platform for this task is the virtual machine. The results are exhibited in **Error! Reference source not found.** and confirm that, for $\alpha = 0.5$, a local performance optimum is reached.

It is important to note that the optimization algorithm had to generate an additional pass with T6 to achieve the target volume when running with $\alpha \in \{0.7, 0.9\}$. Although it does not affect the tool sequence, this supplementary iteration contributes similarly to the increase in computation time required to converge to the desired solution.

The results exhibited in Figure 41 suggest that the virtual machine has the highest computing performance, whereas AWS-GPU presents the slowest platform.

Table 13: Computation time for tool path generation using T5 with different values of α

α	Sweeping time (s)	End volume generation time (s)	Mean total time (s)
0.1	251 +/- 1.2	68+/- 0.3	319.2
0.3	90 +/- 0.7	18 +/-0.2	108.7
0.5	64 +/- 0.5	8 +/- 0.7	72.3
0.7	66 +/- 0.3	7+/-0.1	73.2
0.9	77 +/- 0.6	6 +/- 0.02	83.5

Table 14 shows the relative computation gain of running the algorithm on different platforms compared to the local desktop. Although Tesla M60 floating point performance is 3.7 times as high as Q-M4000 performance, the benchmarking demonstrates a mean gain of only 8.75%. This can be attributed to the use of Citrix XenServer as hypervisor for hardware virtualization which shares the GPU resources between all sessions via a GPU-passthrough (PCI). The specifications of Tesla M60 refer to the entire device and can be fully deployed only if a single user is running the code locally on the platform. Since the performance is related to the number of CUDA of cores allocated for each session, the computation time increases as more sessions are added [59]. Since Tesla M60 is a virtualized resource that is abstracted on a cloud-based platform managed by a group of people, it is hard to allocate the entire GPU to a single user and performance slowdowns are expected.

Using AWS-based GPU shows a slower performance than the local desktop. In fact, the instance g2.2xlarge allows the use of only a single GPU of the K520 grid (2 GPUs). Therefore, the number of associated CUDA cores is reduced to 50%, and so is the

maximum reachable floating-point performance. Additional GPU-overhead is caused by cross-node communication between geographically remote desktops.

Table 14: Computation gain for each platform

Platform	Mean Gain	Maximum reachable floating point performance (TFLOPS)
Local- Quadro M4000		2.573
VM- Tesla M60	+8.75%	9.666
AWS -Grid K520	- 10.23%	2.488

The path-resolution effects are investigated with a step over that is dependent upon voxel size. The results can be found in the Appendix.

5.1.2 Variation of Tool Sequence with Graphic Resolution

In this section, the dependency of tool sequence selection upon the graphic resolution of the voxelized part is investigated. The objective is to determine the impact of a changed voxel size on the generated solutions. To this end, the optimization algorithm is simulated on the candle holder example, but this time with 4 different voxel sizes. The size of the voxel refers to the length of the side of the cubic voxel expressed in mm. With higher voxel resolution, the number of cubic elements forming the part increases and the software has to process more voxels, which overloads the GPU kernels in each instruction. On the other hand, a lower voxel resolution increases the approximation error and may lead to inaccurate tool paths that are inapplicable when dealing with real parts. Since the virtual machine has

shown the highest performance in the previous section, the simulations are run on this platform.

The tooling results for different voxel sizes are summarized in Table 15.

Table 15: Variation of optimal tool sequence with changed voxel size

Voxel size ↓ / Pass index →	1	2	3	4	5	6	7	8	9	10
0.03 mm	T1	T1	T4	T4	T5	T5	T5	T5	T6	T6
0.05 mm	T1	T1	T4	T4	T5	T5	T5	T5	T6	T6
0.07 mm	T1	T1	T3	T4	T5	T5	T5	T6		
0.1 mm	T1	T1	T4	T5	T5	T5	T5	T5		

For voxel side length of 0.03 mm and 0.05 mm, the generated solution is identical. This implies, that opting for a voxel size of 0.05 mm is an optimal choice from a computational perspective, since it eases the GPU burden and converges to the same solution as the configuration with higher resolution. Increasing the voxel size reduces the number of passes required to achieve the target volume as shown in the case of 0.07 mm and 0.1 mm. The discretization error grows with larger voxels and the checking operation of voxel removal is less accurate, since it is based on Boolean operations that verify the intersection of the cutter with the cells on the material boundary layer. As a result, the tool is removing much more material than with smaller voxels. A comparison second and the third case shows that the tool sequence with 0.07 mm voxel size selects T3 before shifting to T4. Similarly, the 4^{th} case suggests a shift to T5 in the 4^{th} pass. The change in the tool sequence can be attributed to the voxel-based approximation of the part geometry and the effects of

lowering the graphic resolution on the offsetting results and the generated tool path, which influences directly the computation of removed volume and consequently the output of the optimization algorithm.

Since the computation time of parallel algorithms depends upon the number of data entities processed by GPU, a relationship between the optimization run time and the number of processed voxels is investigated. It is assumed that the number of voxels is inversely proportional to the unitary voxel volume, since adding up all voxels would approximately lead to the same total volume.

Figure 42 shows a linear relationship between the inverse of single voxel volume and the optimization run time, which confirms the above-mentioned assumptions.

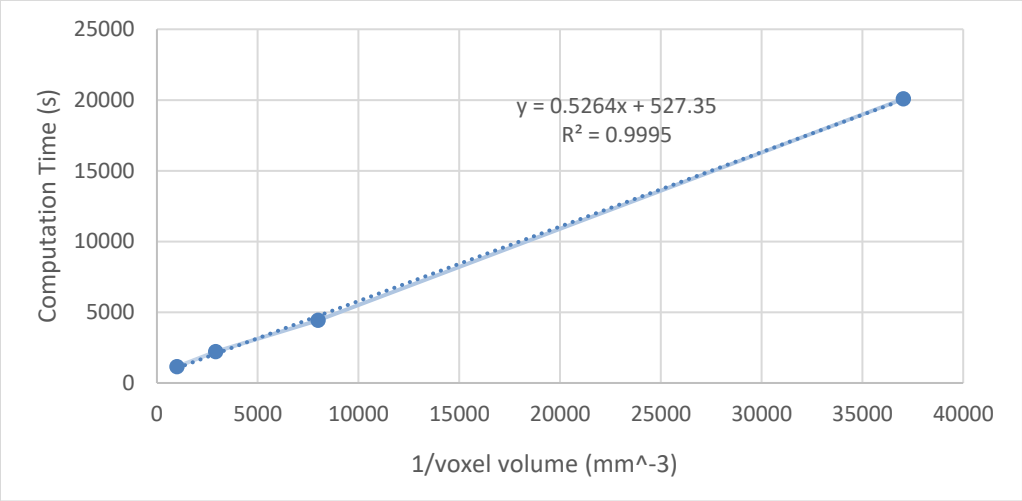


Figure 42: Impact of voxel size on computation time

5.2 Results of AMRR-Based Optimization

The candle holder is the considered part for this section. After voxelization, the part is fed to the optimization algorithm shown in Figure 21. For the current implementation,

the voxel size is 0.05 mm and the step over corresponds to 0.5 of the radius of the deployed tool. Additionally, the accessibility maps are generated with high resolution; this corresponds to a 2°-step size in both theta and phi directions. This configuration, although computationally expensive, allows to overcome the geometric complexity of the part by delivering a more accurate accessibility range, and therefore minimizing the amount of inaccessible points and path sequences. The lower threshold of the path filter is set at 0.4 volume unit/ time unit. This value is chosen based on multiple observations of volume curves and enables not only the elimination of low-efficiency path sequences, but also the downsizing of computational effort for access map generation. An example of a filtered path is given by Figure 43.

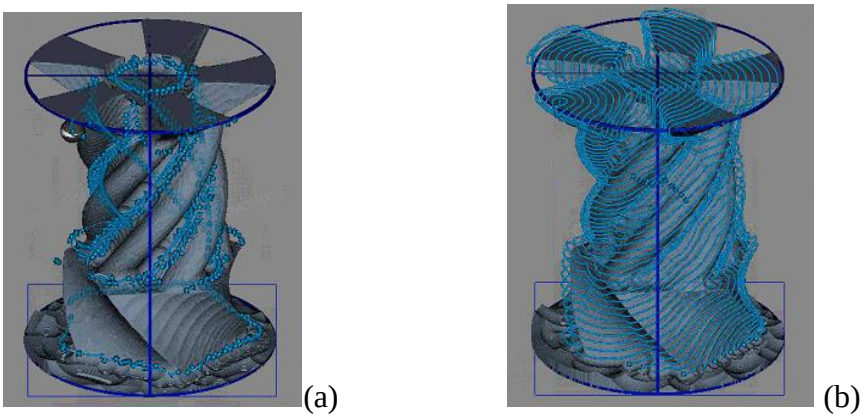


Figure 43: (a) Filtered and (b) unfiltered path for the candle holder example using a lower threshold of 0.4

In each iteration, the algorithm picks the tool with the highest average material removal rate calculated based on the generated velocity profiles and linear curve fitting. A comparison between volume-based and AMRR-based optimization results is shown in Table 16:

Table 16. Variation of optimal tool sequence with changed voxel size

Pass	1	2	3	4	5	6	7	8	9	10
VBO sequence	T1	T1	T4	T4	T5	T5	T5	T5	T6	T6
AMRR sequence	T1	T1	T3	T4	T5	T5	T5	T5	T6	T6

The two first passes are realized by T1, as it presents the tool showing the highest AMRR (950 mm^3/min during the first pass), as exhibited in Figure 44. The unique difference in the generated sequences is found in the 3rd pass, where the VBO strategy opts for T4, whereas the AMRR strategy selects T3. Although the results presented in the previous section suggest that T4 has slightly higher removal volume than T3 during the 3rd pass, an analysis of volume/time curves demonstrates that T3 removes material 62% faster than T4. Figure 45 illustrates this difference in material removal rate. While the final volume is achieved by T3 within 60 min, T4 requires 95 minutes to remove almost the same amount of material. These results certify that AMRR-based optimization is a more efficient strategy for tool selection that prioritizes faster tools and economizes machining time.

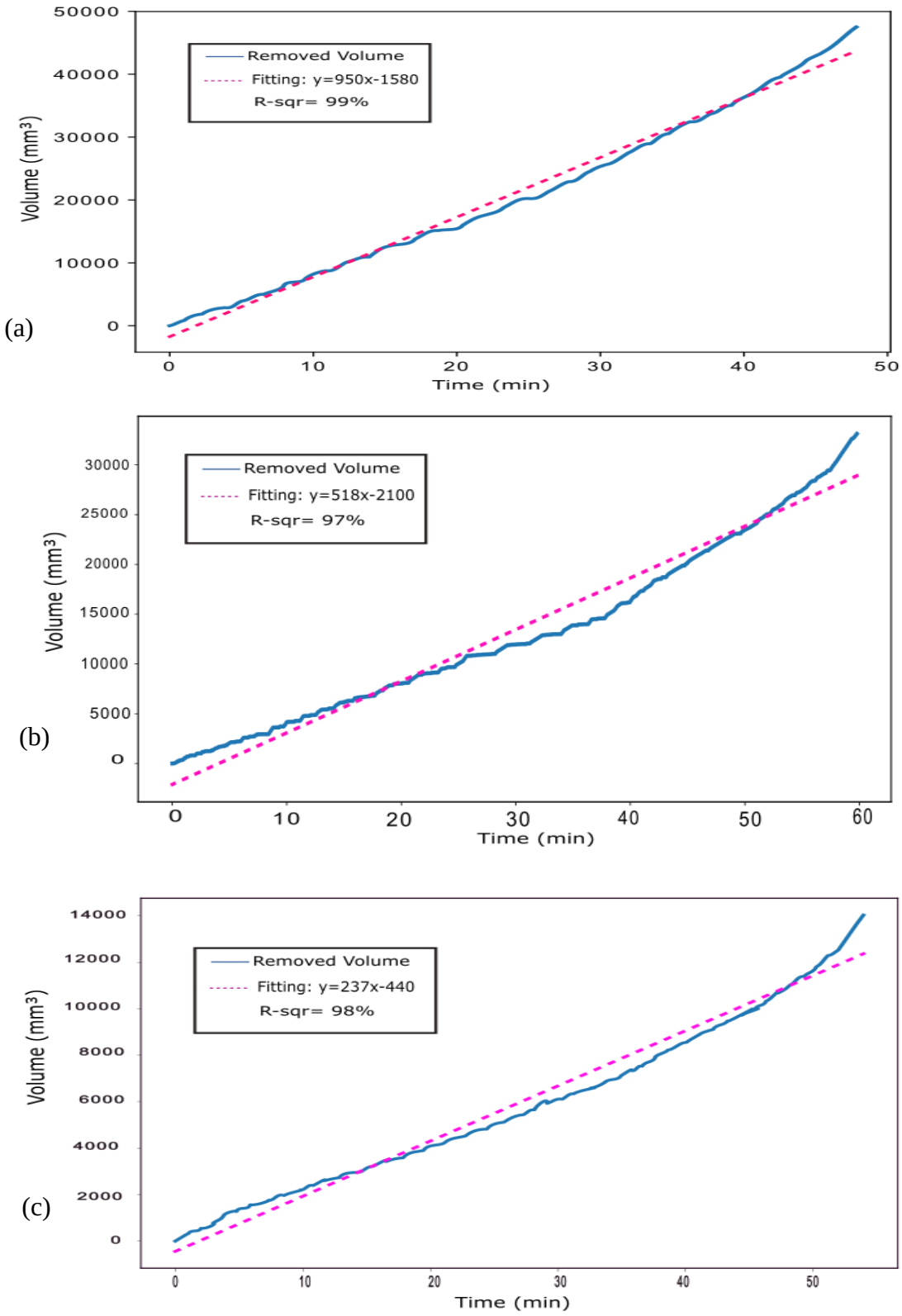


Figure 44: Volume curves of the Tools (a) T1, (b) T2, (c) T3 during the 1st pass

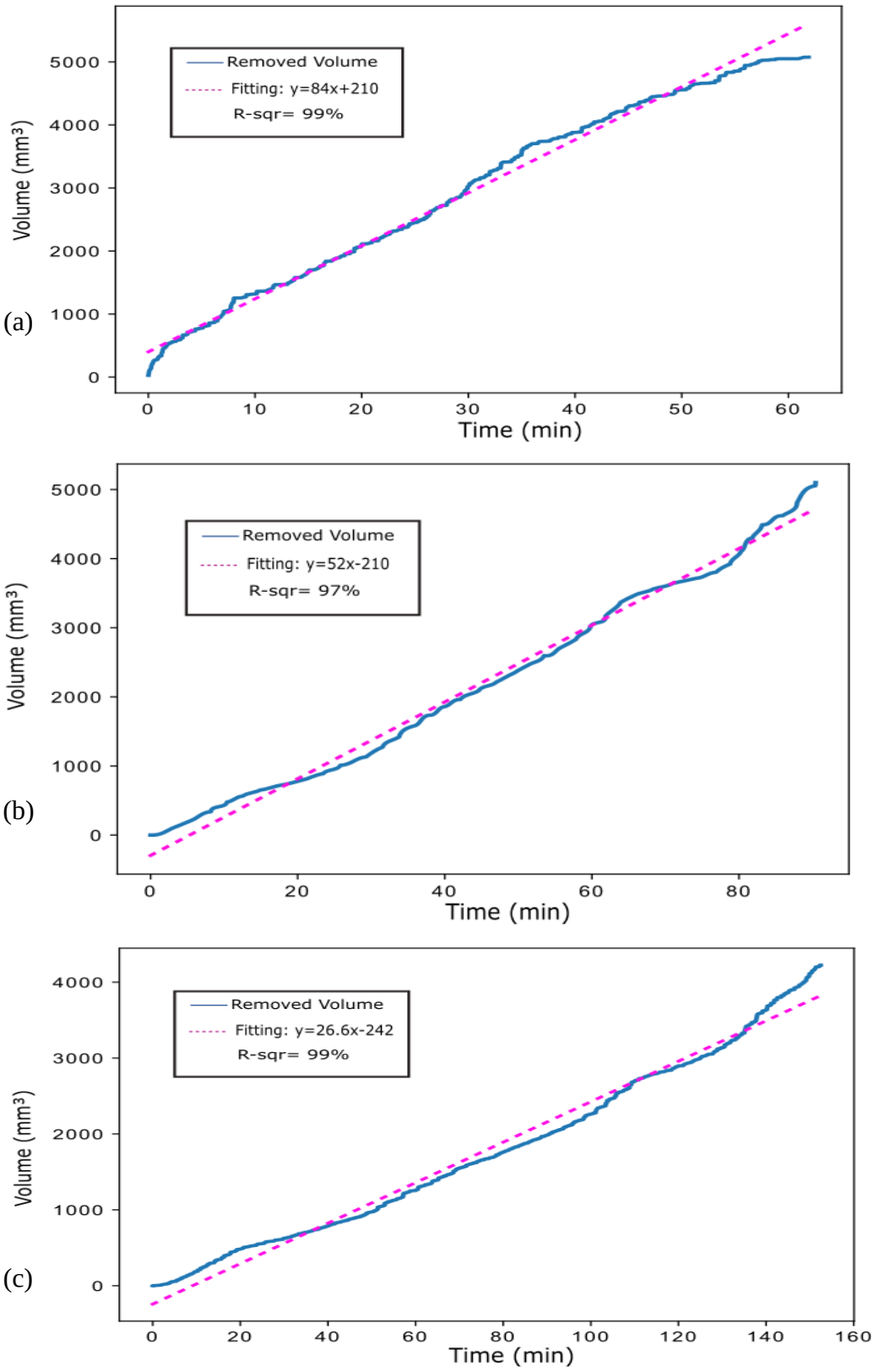


Figure 45: Volume curves of the Tools (a) T3, (b) T4, (c) T5 during the 3rd pass

The currently used algorithm involves the computation of machining time during each pass and is beneficial for the estimation of the time required to produce the entire part on a 5-axis machine. Using the optimal tool sequence, the machine would require 880 minutes (almost 15 hours) to manufacture the candle holder, as exhibited in Figure 46. This approach allows also the manufacturer to evaluate quality-cost-time tradeoffs, as it estimates the time required for surface finishing, and similarly the remaining material after each pass, which is a quality indicator of the end product.

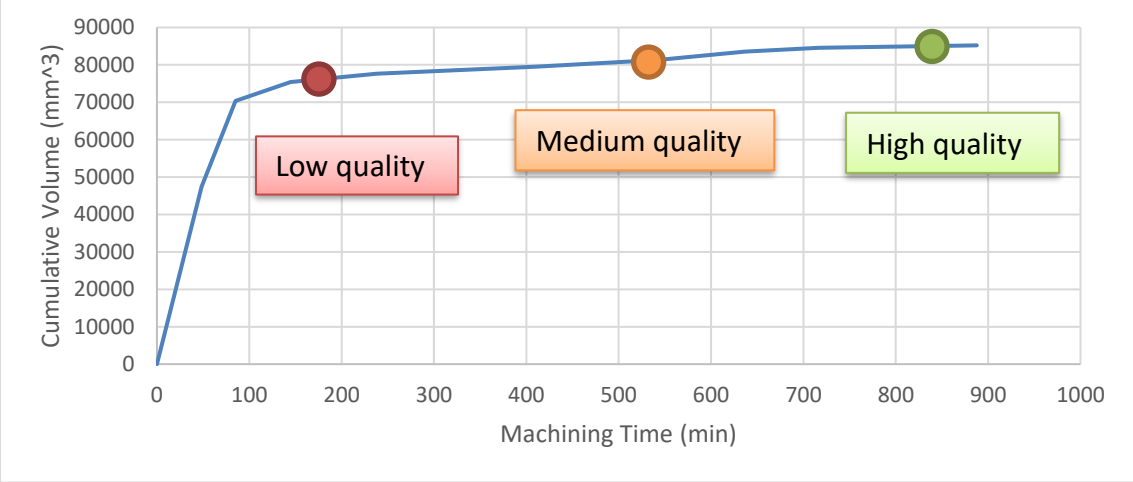


Figure 46: Volume/time curve for producing the candle holder

5.3 Results of Two-Objective Optimization

Based on the results of the two previously introduced optimization concepts, the machine would need 10 passes to achieve the target volume. However, running all possible combinations of tools within 10 passes along the directed acyclic graph presented in 4.3 is a computationally intensive operation. For simplicity's sake, the special case of single-source single-sink DAG is considered. The number of passes is reduced to 5 and the tool selection can be done only among 5 tools (T1, T2,…,T5) . Therefore, the problem is

reduced to a 5x5 grid with possible movements along diagonals. T1 is the source node and T5 is the sink node. An illustration of the described problem is shown on Figure 47.

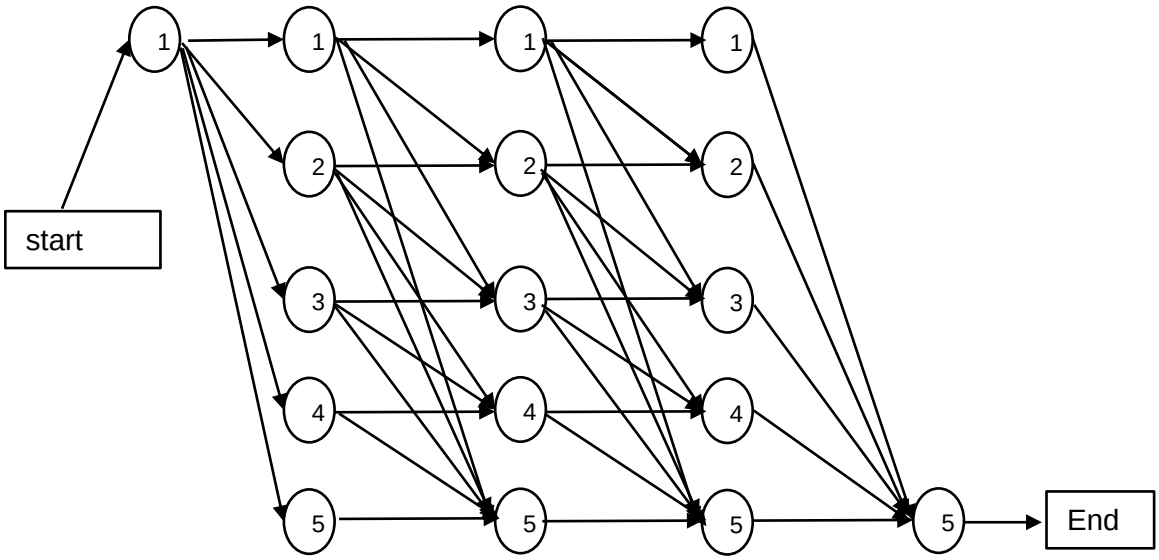


Figure 47: Simplified two-objective tooling problem

The current problem formulation suggests the existence of 35 possible sequences composed of 5 tools that begin with T1 and end with T5 (35= (5+4+3+2+1) +(4+3+2+1) +(3+2+1) +(2+1) +1). With 35 sequences, the algorithm has to run 175 passes including offsetting operations, path generation and filtering. By arranging the sequence order in such a way that only non-redundant subsequences are removed and redundant ones are kept from previous computations, the number of necessary passes is reduced to 87 (reduction of 50% computational load).

After computing all weighted edges between the grid's vertices, the sequence with the highest cumulative volume removal is generated using dynamic programming, followed by the clustering of top 6 sequences. To find the optimal solution, the machining time is

quantified using the path length and feed rate $t = \frac{l}{v_f}$ and computed for the clustered feasible solutions. Table 17 gives the quantified machining time and total removed volume after performing 5 passes for the top 6 sequences. Coincidently, the two optimization criteria were fulfilled by the same solution, which presents, for the example of candle holder, an ideal case. A 2D visualization for all optimization results is introduced in Figure 48, where each solution is represented by an (X, Y) tuple corresponding respectively to the quantified machining time and total volume removed. The 6 feasible solutions are clustered and the optimal case is the member of the selected ensemble showing the lowest X-coordinate. This point is generated through the sequence (T1-T1-T3-T4-T5).

Table 17: Top 6 sequences based on volume removal

Pass index	1	2	3	4	5	Quant. Time (min)	Tot. volume removal (mm^3)
Seq 1	T1	T1	T3	T4	T5	22.95	79,500
Seq 2	T1	T1	T4	T4	T5	24.81	79,277
Seq 3	T1	T1	T2	T4	T5	22.97	79,042
Seq 4	T1	T1	T3	T5	T5	24.45	79,034
Seq 5	T1	T1	T4	T5	T5	26.4	78,916
Seq 6	T1	T2	T2	T4	T5	26.87	78,808

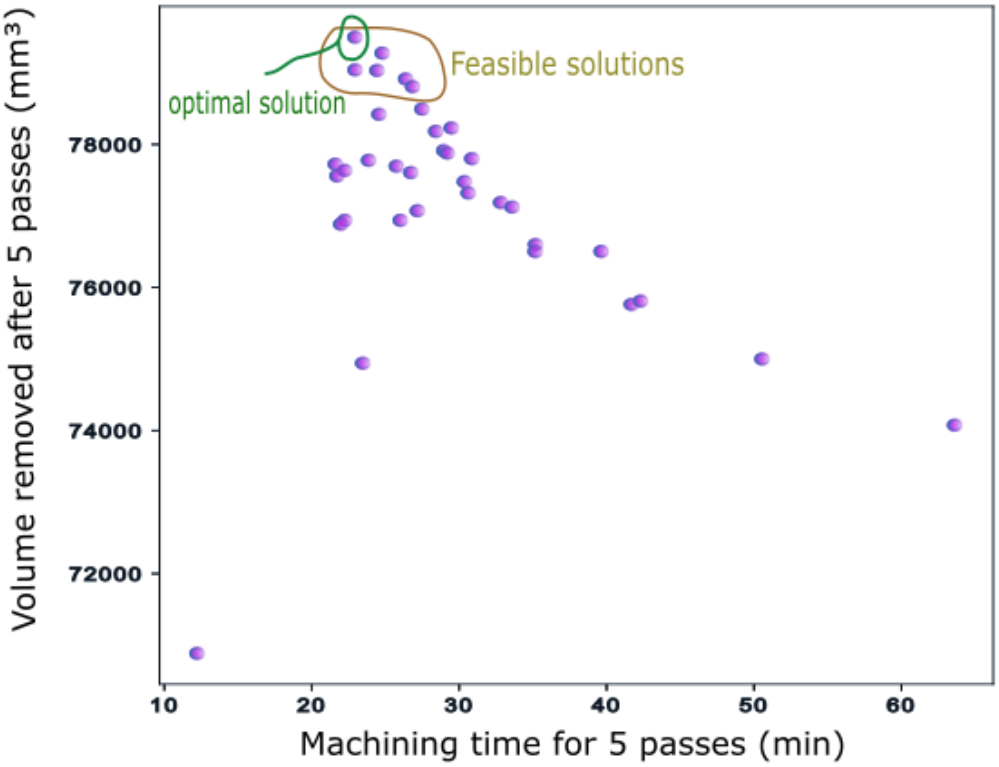


Figure 48: Visualization of two-objective optimization results

CHAPTER 6. DISCUSSION AND CONCLUSIONS

Conclusions

While the 3 formulated algorithms for tool sequence optimization have shown comparable results for the generated sequences, it is important to highlight the advantages and disadvantages of each approach in order to attest its applicability and generalizability on all tool selection problems. Using a volume-based greedy algorithm is a computationally inexpensive approach that delivers acceptable results not only with complex geometries but also with simple-to-machine parts, whose tool sequence can be intuitively obtained. Generally, for this type of parts, running the optimization algorithm is merely a way to validate the manufacturer's choices and not influence them. However, this technique can be beneficial for users with less experience that need to investigate multiple solutions before proceeding to G-code generation. Through a simple implementation, the volume-based strategy carries a large amount of information that would be inaccessible via traditional methods. The machining potential of each tool is assessed during the pass; this includes the engaged depth of cut, the gouging avoidance through offsetting and most importantly the amount of volume removed by the tool, which is hard to estimate using manual calculations and is a computationally intensive task when using parametric

surfaces. Nevertheless, this method delivers only suboptimal solutions and has a non-recursive structing preventing it from reassessing taken decisions. Additionally, it does not guarantee a collision-free path, since the removed volume is calculated while the tool is keeping the same orientation (θ', φ') along the entire generated path. The most relevant disadvantage is, however, the exclusion of time aspects.

The second introduced strategy overcomes this disadvantage by presenting a AMRR-based approach that targets the fastest tools in each pass. This method has shown its efficacy not only in reducing the overall machining time compared to the volume-based strategy, but also in maintaining only the relevant sections of the generated path and ensuring collision-free machining due to the generated accessibility maps. However, running the entire *SculptPrint* workflow for each tool is a computationally intensive process that requires powerful hardware, exclusively dedicated to solving the current problem. The solution took more than 18 hours to be computed, compared to an average of 3 hours for the first strategy. Another drawback is related the linearization of volume curves. Although, the curve fitting showed very low regression error (high r-squared), it is not guaranteed to have the same behavior with all possible geometries, especially with a path filter that is based on multiple observations and is not preliminarily defined. Using a greedy algorithm as main paradigm, the method can also lead to suboptimal solutions.

To tackle this problematic, a two-objective optimization was formulated. The method is based on a selection tree method that can be formulated as directed acyclic graph. With a simplified example reducing the complexity to 5x5 grid, the results show that the two first optimal solutions correspond to those of AMRR and volume-based strategies. Despite its holistic approach, the two-objective strategy can be a time-consuming process that do not

necessarily lead to significant advantages compared to the two aforementioned strategies. Furthermore, the quantification of machining time may be inaccurate, since the feed rate of the considered tools can be continuously adjusted throughout the machining process.

This thesis presented a novel voxel-based approach to optimize the tool sequence selection during the manufacturing of complex parts using a 5-axis CNC machine. The implementation was supported by SculptPrint; a voxelized CAM software that performs GPU-enabled volume offsetting and tool path generation. 3 tooling strategies were proposed with different optimization criteria. The results of each strategy were visualized and compared. To validate the robustness of this approach, a part with simple geometry was considered and its corresponding tool sequence was generated and showed high compatibility with traditional methods. To compare GPU performance and efficiency, 3 platforms were investigated, where 2 platforms possess virtualized GPU and one platform has local GPU. The results showed that virtualized Tesla M60 has the highest computing performance, although the computation time doesn't meet the expectations placed on the hardware. A linear relationship between computation time and the number of voxels was also deduced through variation of voxel size and iterative running of the optimization algorithm. The variation of path resolution affected similarly the generated tool sequences. The advantages and disadvantages of each optimization strategy were discussed, the limitations of the study from software and hardware perspective were given, and areas for future exploration based on the outcome of this study were presented.

Contributions of the Thesis

1- This study presents a voxel-based approach to solve the tool selection problem. The main advantage of this approach is the ability to graphically approximate all types of free form surfaces through 3D arrays, whose computation can be executed on parallel platforms. The voxelization enables also the visualization of different stages of the machining process and optimize the parameters of the tool selection process based on volume, efficiency, and quality criteria
2- The formulated algorithms cover the tool selection for the process planning of the entire part. Starting with a stock, a tool is selected at each stage to perform a pass and brings the part one step closer from the target shape.
3- The parallelization of computing operations realizable due to the voxel-based formulation allows the use of GPU as parallel computing platform. This enables the reduction of computational load that is generally a major issue in CPU-enabled tool selection algorithms developed with parametric surfaces [17].
4- The thesis presents an example of cloud manufacturing applications and investigates the performance of different platforms. Amazon Web Service (AWS) is used as computing infrastructure to host manufacturing simulations and grant access to virtualized hardware.

Study Limitations

1- Software limitations: The results of the thesis are obtained within the actual development framework of *SculptPrint.* The optimization algorithms are developed using only the available python commands. Additionally, it is impossible to modify the automated tool path generation, unless a manual/ human intervention is considered. However, the goal of the thesis is to formulate an automated tool selection process

without requiring any manual input during the execution of the code. Additionally, most of *SculptPrint* functionalities are not applicable in the case of turning operations, which shifted the focus of the thesis on milling tools.

2- Hardware Limitations: Although the voxel-based modeling, offsetting and path generation are highly parallelizable operations, their computing efficiency is heavily dependent on the hardware performance: the more CUDA cores are dedicated to a specific job, the faster the execution can get. Sharing the GPU resources between multiple users leads to performance slowdowns. To run the optimization algorithm on a virtualized platform, it is necessary to make sure that the GPU is currently not overloaded and that only few users are sharing the corresponding CUDA cores. It is also necessary to set a certain timeout threshold for the TDR (timeout detection and recovery) functionality to prevent the operating system from enforcing the termination of CUDA driver, if the execution of an instruction has exceeded the defined threshold.

3- The formulated algorithms consider only virtual material with unspecified mechanical or chemical properties; no influence of material on machining parameters was investigated.

4- The generated velocity profiles in *SculptPrint* are not necessarily reproducible in real machine environment. The machining time depends on the speed with which the machine executes incrementally the given G-code and experiments have shown high discrepancy between the expected and actual machining time.

Future Recommendations

1- Improvements on the software workflow and performance can be investigated. The automated tool path generation should be enhanced by focusing the path on the regions showing more material left than those less machinable. This improvement could reduce the computation time required to sweep and consolidate the voxel elements and filter the path.

2- Another area of exploration is the GPU-resource-sharing and management through virtualization. Alternatives for prioritizing particular CUDA calls and allocating more cores for one user can be investigated. AWS cloud-based instances can be used to enhance the computing performance, by choosing those with more powerful GPU; This is, however, associated with higher service costs that can also be optimized. There is no doubt that the price per TFLOP will decrease in the next few years, and the use of virtualized GPGPU for CAM applications will gain more attractiveness [59].

3- The algorithms developed in this thesis can be extended to treat different materials and optimize the choice of machining parameters. The choice of step over can be made based on the desired scallop height and the mechanical properties of both cutter and stock material. One other area of exploration is the inclusion of tool wear models in tool selection process. The algorithm can be modified to involve tool life, tooling costs and set up time.

4- The tool selection strategy can be embedded in the company's supply chain management and production planning by forecasting machining costs, scheduling available resources, and managing orders for tools and materials.

APPENDIX A.

A.1 Python Script for Volume-Based Optimization

```
import time
import sculptprint
import numpy
#start = time.time()
#stopcon=input("if you want to break, please press b")
#end = time.time()

#elapsed = end - start
#print(elapsed)
#if stopcon.strip()=='b':
#         print(1)
stepover= [0.1, 0.2, 0.4, 0.6]#[ 0.03, 0.07,  0.1]#[0.1, 0.3, 0.5, 0.7, 0.9]
ll=0
timeeach=[0]*int(len(stepover))
while ll<int(len(stepover)):
      start = time.time()
      sculptprint.start()
      sculptprint.new()
      sculptprint.open(r'C:\Users\aameur3\Desktop\pythsculp.scpr')
      #sculptprint.create_volumes(stepover[ll], 12.700000)

      toolcatalog=[9, 7, 12, 10, 8, 11]
      depthofcut=[1/2*25.4/2,    5/16*25.4/2,    1/4*25.4/2,    3/16*25.4/2,    1/8*25.4/2,
3/32*25.4/2]
      print (depthofcut)
      stockvolume=294160
      partvolume=258722
      Vtr=stockvolume-partvolume
      TVr=0
      i=1
      j=1
      N=len(toolcatalog)
      Vrp= numpy.zeros(shape=(N,20))
      Toolseq=numpy.zeros(shape=(1,20))
      while  (j<11):   #(TVr< 0.98*Vtr) and
            while (i<N+1):
                  sculptprint.set_active_tool(str(toolcatalog[i-1]))
                  #mypass=sculptprint.create_pass()
                  #mypass.build_volumes(0, 0.95*depthofcut[i-1],1)
```

```
                sculptprint.auto_generate_tool_pass(0, 0.95*depthofcut[i-1],
0.95*depthofcut[i-1]*0.3*0+stepover[ll])
                print('hello world!')
                passnumber='Tool Pass #'+str(j)
                #sculptprint.set_active_pass(passnumber)
                #myeditor=sculptprint.create_pass_editor()
                #myeditor.filter_pass(0.1,0,0,0,0)
                #sculptprint.finalize_pass_editor(myeditor,0)
                sculptprint.set_active_pass(passnumber)
                Vrp[i-1][j-1]=sculptprint.compute_active_pass_volume_removed()
                print(Vrp[i-1][j-1])
                sculptprint.remove_tool_pass()
                i=i+1
            Vr=numpy.amax(Vrp,axis=0)[j-1]
            i=numpy.argmax(Vrp,axis=0)[j-1]+1
            #if i==1 and j>10 :
                #break
            sculptprint.set_active_tool(str(toolcatalog[i-1]))
            #mypass=sculptprint.create_pass()
            #mypass.build_volumes(0, 0.95*depthofcut[i-1],0.1)
            sculptprint.auto_generate_tool_pass(0, 0.95*depthofcut[i-1], 0.95*depthofcut[i-
1]*0.3*0+stepover[ll])
            Toolseq[0][j-1]=i
            TVr=TVr+Vr
            print(Vrp)
            print(TVr)
            print(Toolseq)
            print(i)
            j=j+1
            #i=1
            #if j==16:
                #stopcon=input("if you want to break, please press b")
                #if stopcon.strip()=='b':
                    #break
    print(Toolseq)
    #numpy.save('Vrp_p'+str(ll),Vrp)
    filename=r'C:\Users\aameur3\Desktop\trycandle01stepover'+str(ll)
    sculptprint.save_as(filename+'_1.scpr')
    sculptprint.stop()
    end = time.time()
    elapsed=end-start
    timeeach[ll]=elapsed
    ll=ll+1
    print(timeeach)
#numpy.save('timeeach2',timeeach)
```

A.2 Python Script for AMRR-Based Optimization

```
import sculptprint
import numpy
#import matplotlib.pyplot as plt
sculptprint.start()
sculptprint.new()
sculptprint.open(r'C:\Users\aameur3\Desktop\simplepart.scpr')

toolcatalog=[9, 7, 12, 10, 8, 11]
depthofcut=[1/2*25.4/2, 5/16*25.4/2, 1/4*25.4/2, 3/16*25.4/2, 1/8*25.4/2, 3/32*25.4/2]
print (depthofcut)
stockvolume=294160
partvolume=258722
Vtr=stockvolume-partvolume
TVr=0
i=1
j=1
N=len(toolcatalog)
Vrp= numpy.zeros(shape=(N,20))
MRRp= numpy.zeros(shape=(N,20))
Toolseq=numpy.zeros(shape=(1,20))


while  (j<10):
      while (i<N+1):
         sculptprint.set_active_tool(str(toolcatalog[i-1]))
         #mypass=sculptprint.create_pass()
         #mypass.build_volumes(0, 0.95*depthofcut[i-1],1)
         sculptprint.auto_generate_tool_pass(0,   0.95*depthofcut[i-1],   0.95*depthofcut[i-
1]*0.5)   #generate pass with tool index i
         print('hello world!')  #misch
         passnumber='Tool Pass #'+str(j)    #string with pass number
         sculptprint.set_active_pass(passnumber)
         Vrp[i-1][j-1]=sculptprint.compute_active_pass_volume_removed()      #  compute
volume of pass
         print(Vrp[i-1][j-1])
         sculptprint.set_active_pass(passnumber)
         myeditor = sculptprint.create_pass_editor()          #filter pass
         myeditor.filter_pass(0.40000, 0.000000, 0, 0, 0)    # mRR threshold 0.4
         sculptprint.finalize_pass_editor(myeditor, 0)
         sculptprint.set_active_pass(passnumber)
         myeditor = sculptprint.create_pass_editor()      #connections
         myeditor.create_connections(1, 0.14980000, 7.500000, 1, 1, 0, -1)
         sculptprint.finalize_pass_editor(myeditor, 0)
         sculptprint.set_active_pass(passnumber)
```

```
        myaccessmaps = sculptprint.create_access_maps_editor()     #access maps high
resolution
        myaccessmaps.set_min_phi(0)
        myaccessmaps.set_theta_resolution(180)
        myaccessmaps.set_phi_resolution(90)
        myaccessmaps.set_voxel_step(5)
        myaccessmaps.generate_all_maps()
        sculptprint.finalize_access_maps_editor(myaccessmaps)
        sculptprint.set_active_pass(passnumber)
        sculptprint.remove_inaccessible_points(2.000000)     #remove inaccessible
        sculptprint.set_active_pass(passnumber)
        myaccesspath = sculptprint.create_access_path_editor()   #tool orientations
        myaccesspath.set_strategy(r'')
        myaccesspath.create_orientations()
        sculptprint.finalize_access_path_editor(myaccesspath)
        sculptprint.set_active_pass(passnumber)
        sculptprint.create_velocity_profile(2000.000000,        400.000000,        6.283185,
10000.000000, 62.831856, 12.700000, 1, 0)   # create velocity profile
        sculptprint.set_active_pass(passnumber)
        dataset=sculptprint.get_active_pass_data(0.1,True)
        k=0
        time= [0]*int(len(dataset)/7)
        vol=  [0]*int(len(dataset)/7)
        y= [0]*int(len(dataset)/7)
        while k<  int(len(dataset)/7):
           time[k]=dataset[7*k+5]
           vol[k]=dataset[7*k+6]
           k=k+1
        print(time)
        print(vol)
        z=numpy.polyfit(time,vol,1)
        c=0
        while c< int(len(dataset)/7):
           y[c]=time[c]*z[0]+z[1]
           c=c+1
        #plt.plot(time,vol)
        #plt.plot(time,y)
        #plt.show()
        print(z[0])
        MRRp[i-1][j-1]=z[0]
        sculptprint.set_active_pass(passnumber)
        sculptprint.remove_tool_pass()
        i=i+1
    MRR=numpy.amax(MRRp,axis=0)[j-1]
    i=numpy.argmax(MRRp,axis=0)[j-1]+1
    Vr=Vrp[i-1][j-1]
```

```
        sculptprint.set_active_tool(str(toolcatalog[i-1]))
    #mypass=sculptprint.create_pass()
    #mypass.build_volumes(0, 0.95*depthofcut[i-1],0.1)
        sculptprint.auto_generate_tool_pass(0,     0.95*depthofcut[i-1],     0.95*depthofcut[i-
1]*0.5)
        sculptprint.set_active_pass(passnumber)
        sculptprint.set_active_pass(passnumber)
        myeditor = sculptprint.create_pass_editor()
        myeditor.filter_pass(0.400000, 0.000000, 0, 0, 0)
        sculptprint.finalize_pass_editor(myeditor, 0)
        sculptprint.set_active_pass(passnumber)
        myeditor = sculptprint.create_pass_editor()
        myeditor.create_connections(1, 0.14985, 7.500000, 1, 1, 0, -1)
        sculptprint.finalize_pass_editor(myeditor, 0)
        sculptprint.set_active_pass(passnumber)
        myaccessmaps = sculptprint.create_access_maps_editor()
        myaccessmaps.set_min_phi(0)
        myaccessmaps.set_theta_resolution(180)
        myaccessmaps.set_phi_resolution(90)
        myaccessmaps.set_voxel_step(5)
        myaccessmaps.generate_all_maps()
        sculptprint.finalize_access_maps_editor(myaccessmaps)
        sculptprint.set_active_pass(passnumber)
        sculptprint.remove_inaccessible_points(2.000000)
        sculptprint.set_active_pass(passnumber)
        myaccesspath = sculptprint.create_access_path_editor()
        myaccesspath.set_strategy(r'')
        myaccesspath.create_orientations()
        sculptprint.finalize_access_path_editor(myaccesspath)
        Toolseq[0][j-1]=i
        TVr=TVr+Vr
        print(Vrp)
        print(TVr)
        print(Toolseq)
        print(i)
        print(MRRp)
        j=j+1
        #i=1
        sculptprint.save_as(r'C:\Users\aameur3\Desktop\candleMRR.scpr')
sculptprint.save_as(r'C:\Users\aameur3\Desktop\candleMRR.scpr')
sculptprint.stop()
```

A.3 Python Script for Multi-Objective Optimization

```
c=1
a=0
import sculptprint
import numpy
def createpass(toolindex,passindex):
   toolcatalog=[9, 7, 12, 10, 8, 11]
   depthofcut=[1/2*25.4/2,  5/16*25.4/2,  1/4*25.4/2,  3/16*25.4/2,  1/8*25.4/2,
3/32*25.4/2]
   sculptprint.set_active_tool(str(toolcatalog[toolindex-1]))
   sculptprint.auto_generate_tool_pass(0,                    0.95*depthofcut[toolindex-1],
0.95*depthofcut[toolindex-1]*0.5)
   passnumber='Tool Pass #'+str(passindex)
   sculptprint.set_active_pass(passnumber)
   myeditor = sculptprint.create_pass_editor()          #filter pass
   myeditor.filter_pass(0.40000, 0.000000, 0, 0, 0)   # mRR threshold 0.4
   sculptprint.finalize_pass_editor(myeditor, 0)
   sculptprint.set_active_pass(passnumber)
   volume=sculptprint.compute_active_pass_volume_removed()
   distance=sculptprint.compute_active_pass_distance()
   return (volume,distance)
sculptprint.start()
sculptprint.new()
sculptprint.open(r'C:\Users\aameur3\Desktop\pythsculp.scpr')

toolcatalog=[9, 7, 12, 10, 8, 11]
depthofcut=[1/2*25.4/2, 5/16*25.4/2, 1/4*25.4/2, 3/16*25.4/2, 1/8*25.4/2, 3/32*25.4/2]
Vrp= numpy.zeros(shape=(36,15))
dd=[1,1,1,1,1]

for i in range(1,6):
   (Vrp[0][4+i],Vrp[0][9+i])=createpass(1,i)
print(Vrp)


for j in range(1, 6):
   for k in range(j, 6):
      for l in range(k,6):
         cc=[1,j,k,l,5]
         CCC = [a - b for a, b in zip(cc, dd)]
         indexpass=next((i for i, x in enumerate(CCC) if x), None)
         passnumber1='Tool Pass #'+str(indexpass+1)
         sculptprint.set_active_pass(passnumber1)
         sculptprint.remove_tool_pass()
         c=c+1
```

```
        (Vrp[c][5:indexpass+4],Vrp[c][10:9+indexpass])=(Vrp[c-
1][5:indexpass+4],Vrp[c-1][10:9+indexpass])
        for counter in range(indexpass+1,6):
            (Vrp[c][4+counter],Vrp[c][9+counter])=createpass(cc[counter],counter)
            print(Vrp)
        dd=[1,j,k,l,5]
print(a)
print(c)
print(Vrp)
```

A.4 Results for Voxel-dependent Step Over

Step over/ voxel size	Pass1	Pass2	Pass3	Pass4	Pass5	Pass6	Pass7	Pass8	Pass9	Pass10
2	T1	T1	T4	T4	T5	T5	T5	T5	T6	T6
4	T1	T1	T4	T4	T5	T5	T5	T5	T6	T6
6	T1	T1	T4	T4	T5	T5	T5	T5	T6	T6

REFERENCES

[1] T. Lim, J. Corney and M. Ritchie, "Optimizing tool selection," *International Journal of Production Research,* vol. 39, no. 6, pp. 1239-1256, 2001.

[2] C. Chiou and Y. Lee, "A machining potential field approach to tool path," *Computer-Aided,* vol. 34, no. 5, pp. 357-371, 2002.

[3] D. Veeramani and Y.-S. Gau, "Analytical models for optimal NC machining of regular convex polygonal pockets," *International Journal for Production, vol. 35, no. 9, pp. 2621-37, 1997.*.

[4] Z. Chen and Q. Fu, "An optimal approach to multiple tool selection and their numerical control path generation for aggressive rough machining of pockets with free-form boundaries," *Computer-aided Design,* vol. 43, pp. 651-63, 2011.

[5] A. Jonjea, Y. Weifeng and Y.-S. Lee, "Greedy tool heuristic approach to rough milling of complex shaped pockets," *IIE Transactions* , vol. 35:10, pp. 953-963, 2003.

[6] I. Nadjakova and S. Mcmains, "Finding an Optimal Set of Cutter Radii for 2D Pocket Machining," *ASME Proceedings | Manufacturing Engineering,* pp. 351-358, 2004.

[7] H. Ramaswami, R. Shaw and S. Anand, "Selection of optimal set of cutting tools for machining of polygonal pockets with islands," *The International Journal of Advanced Manufacturing Technology,* vol. 53, pp. 963-977, 2011.

[8] D. Mount, S. Arya and S.-W. Cheng, "Approximation Algorithm for multiple tool milling," *International journal of computational geometry & applications,* vol. 11, pp. 339-372, 2001.

[9] T. Lim, J. Corney and D. Clark, "Exact Tool Sizing for Feature Accessibility," *International Journal of Advanced Manufacturing Technology,* vol. 16, pp. 791-802, 2000.

[10] R. D' Souza, P. Wright and C. Sequin, "Automated microplanning for 2.5D Pocket machining," *Journal of Mfg Systems,* vol. 20, pp. 228-96.

[11] R. M. D'Souza, P. K. Wright and C. Séquin, "Handling Tool Holder Collision in Optimal Tool Sequence Selection for 2.5-D Pocket Machining," *ASME 2002 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference,* vol. 22, pp. 553-559, 2002.

[12] Z. Ahmad, K. Rahmani and R. M. D'Souza, "pplications of genetic algorithms in process planning: tool sequence selection for 2.5-axis pocket machining".

[13] A. W. Churchill, P. Husbands and A. Philippides, "Multi-objective tool sequence and parameter optimization for rough milling applications," *2013 IEEE Congress on Evolutionary Computation,* pp. 1475-82, 2013.

[14] G. Elber, "Freeform surface region optimization for 3-axis and 5-axis milling," *Computer-Aided Design,* vol. 27, pp. 465-470, 1995.

[15] Y.-S. LEE and T.-C. CHANG, "Automatic cutter selection for 5-axis sculptured surface machining," *International Journal of Production Research ,* Vols. 998-1002, p. 34, 1996.

[16] C. Jensen, W. Red and J. Pi, "Tool selection for five-axis curvature matched," *Computer-Aided Design,* vol. 34, no. 3, pp. 251-266, 2002.

[17] L. Li, "Process planning for five-axis milling of sculptured surfaces," PhD Thesis, National University of Singapore, 2007.

[18] L. Hayan, "Process Planning optimization for five-axis sculpture surface finishing," PhD thesis, National University of Singapore, 2012.

[19] J. Tarbutton, "Automated Digital Machining for Parallel Processors," PhD Dissertation, Clemson University, 2011.

[20] Y. Lee, "Non-isoparametric tool path planning by machining strip evaluation," *Computer-Aided Design,* vol. 30, no. 7, pp. 559-570, 1998.

[21] H. Li and H. Feng, "Efficient five-axis machining of free-form surfaces with," *International Journal of Production,* vol. 42, pp. 2403-2417., 2004.

[22] S. Park and P. Choi, "Tool-path planning for direction-parallel area milling," *Computer-aided Design,* vol. 32, pp. 17-25, 2000.

[23] S. Park, "Tool-path generation for Z-constant contour machining," *Computer-Aided Design,* vol. 35, no. 1, pp. 27-36, 2003.

[24] G. Loney and T. Ozsoy, "NC machining of free form surfaces," *Computer-Aided Design,* vol. 19, no. 2, pp. 85-90, 1987.

[25] J. Hwang, "Interference-free tool-path generation in the NC machining of parametric compound surfaces," *Computer-Aided Design,* vol. 24, no. 12, pp. 667-676, 1992.

[26] S. Yuwen, G. Dongming and J. Zhenyuan, "Iso-parametric tool path generation from triangular meshes for free-form surface machining," *The International Journal of Advanced Manufacturing Technology,* vol. 28, no. 7, pp. 721-726, 2006.

[27] G. Elber and E. Cohen, "Tool path generation for free form surface model," in *ACM symposium on Solid modeling and applications*, 1993.

[28] S. Park and B. Choi, "Tool-path planning for directio-parallel area milling," *Computer-Aided Design,* vol. 32, no. 1, pp. 17-25, 2000.

[29] S. e. a. Ding, "Adaptive iso-planar tool path generation for machining of freeform," *Computer-Aided Design,* vol. 35, no. 2, pp. 141-153, 2003.

[30] C. Tournier, "Iso-scallop tool path generation in 5-axis milling," *The International Journal of Advanced Manufacturing Technology,* vol. 25, no. 9, pp. 867-875, 2005.

[31] C. Tournier and C. Lartigue, "5-axis Iso-scallop Tool Paths along Parallel Planes," *Computer-Aided Design and Application,* vol. 5, no. 4, 2008.

[32] S. Li and B. Jerard, "5-axis machining of sculptured surfaces with a flat-end," *Computer-Aided Design,* vol. 26, no. 6, pp. 165-178, 1994.

[33] J. Duncan and S. Mair, Sculptured Surfaces in Engineering and Medicine, University of Cambridge Press, 1983.

[34] C. Matthew, "Implementing Rapid Prototyping Using CNC Machining (CNC-RP) Through a CAD/CAM Interface," in *Proceedings of the Solid Freeform Fabrication Symposium*, Austin, TX, 2007.

[35] C. Jun and D. Kim, "A new curve-based approach to polyhedral machining," *Computer-Aided Design,* vol. 34, pp. 379-389, 2002.

[36] B. Choi, "Compound surface modelling and machining," *Computer-Aided Design,* vol. 20, no. 3, pp. 127-138, 1988.

[37] B. Choi and R. Jerad, "C-Space approach to tool-path generation for die and mold machning," *Computer-Aided Design,* vol. 29, no. 9, pp. 657-669, 1997.

[38] M. Innui, "Fast Inverse offset computation using polygon rendering hardware," *Computer-Aided Design,* vol. 35, no. 2, 2003.

[39] Y. Ren, H. Tzong Y and Y. Lee, "Clean-up tool path generation by contraction tool method for machining complex polyhedral models," *Computers in Industry,* vol. 54, no. 1, pp. 17-33, 2004.

[40] K. Morishige, Y. Takeuchi and K. and Kase, "Tool Path Generation using CSpace for 5-axis control machining," *Journal of manufacturing science and engineering,* vol. 121, no. 1, 1999.

[41] J. Fan and A. Ball, "Quadric Method for Cutter Orientation in Five-Axis Sculptured Surface Machining," *International Journal of Machine Tools and Manufacture,* vol. 48, no. 7, pp. 788-801.

[42] P. Gray, J. Ismail and S. Bedi, "Graphics-Assisted Rolling Ball Method for 5-Axis Surface Machining," *Computer-Aided Design,* vol. 36, no. 7, pp. 653-663.

[43] P. Gray, F. Ismail and S. Bedi, "Rolling Ball Method for 5-axis Surface Machining," *Computer-Aided Design,* vol. 35, no. 4, pp. 347-357.

[44] D. Konobrytskyi, "Automated CNC Tool path planning and machining simulation on highly parallel processors," Clemson University, PhD Dissertations, 2013.

[45] J. Fung and F. Tang, "Mediated Reality using Computer Graphics Hardware for Computer Vision," *Meditated Reality using Computer Graphics Hardware for Computer Vision,* vol. 7, no. 3, 2002.

[46] nVidia, "nVIDIA. GPU Acclerated Research," nVidia, 2009. [Online]. Available: http://www.nvidia.com/object/cuda_home.html#.. [Accessed 20 6 2017].

[47] N. Galoppo, N. Govindaraju and M. Henson, "Efficient Algorithms for Solving Dense Linear Systems on Graphics Hardware," in *SC '05 Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, 2005.

[48] M. Rumpf, "Level set segmentation in graphics hardware," in *Image Processing, 2001. Proceedings. 2001 International Conference on, Volume: 3*, 2001.

[49] A. Haider, A. Abdelfattah and S. Tomov, "High-Performance Cholesky factorization for GPU-only Execution," University of Tennessee, 2015.

[50] L. Crivelli and M. Dunbar, "Evolving Use of GPU for Dassault Systems Simulation Products," nVidia GPU Tech Conference, 2012.

[51] K. Group, "OpenCL - the Open Standard for Parallel Programming of Heterogenous Systems," 2012.

[52] nVIDIA, "Parallel Programming and Computing platform| CUDA," 2012.

[53] R. Khardekar, G. Burton and S. Mcmains, "Finding feasible mold parting directions using graphics hardware," *Computer-Aided Design,* vol. 38, no. 4, pp. 327-341, 2006.

[54] A. Krishnamurthy, "Parallel GPU Algorithms for Mechanical CAD," PhD Dissertation, University of California Berkeley, 2010.

[55] D. Roth and F. Ismail, "Mechanistic modelling of the milling process using an adaptive depth buffer," *Computer-Aided Design,* vol. 35, no. 14, pp. 1141-1152, 2003.

[56] B. Tukora and T. Szalay, "Real-time determination of cutting force coefficients without cutting geometry restriction," *International Journal of Machine Tools and Manufacture,* vol. 51, no. 12, 2011.

[57] H. Hsieh and c. Chu, "Particle swarm optimisation (PSO)-based tool path planning for 5-axis flank milling accelerated by graphics processing unit (GPU)," *International Journal of Computer Integrated Manufacturing* , vol. 24, no. 7, 2011.

[58] Citrix, "Virtualizing the design engineer workstation is achievable today!," Citrix, 2012. [Online]. Available: https://www.citrix.com/blogs/2012/10/16/virtualizing-the-design-engineer-workstation-is-achievable-today/. [Accessed 22 6 2017].

[59] R. Lynn, T. Tucker, M. Hossain and T. Kurfess, "Voxel Model Surface Offsetting for CNC Toolpath Generation using Virtualized High-Performance Computing," *Journal of Manufacturing Systems,* vol. 43, no. 2, pp. 296-304, 2017.

[60] G. Popek and R. Goldbert, "Formal requirements for virtualizable third generation architectures," *Communications of the ACM,* vol. 17, no. 7, pp. 412-121, 1974.

[61] L. Shin, H. Chen and J. Sun, "vCUDA: GPU-Accelerated High-Performance Computing in Virtual Machines," *IEEE Transactions on COmputers,* vol. 61, no. 6, pp. 804-816, 2012.

[62] V. Gupta and A. Gavrilovska, "GViM: GPU-accelerated Virtual Machine," *ACM,* pp. 17-24, 2009.

[63] J. Konobritskyi, T. Tucker and T. Kurfess, "Hybrid Dynamic Tree Structure and Accessibility Mapping for CNC Machining Path Planning," 2015.

[64] M. Hossain, R. Vuduc, C. Nath and T. Kurfess, "A Graphical Approach for Freeform Surface Offsetting With GPU Acceleration for Subtractive 3D Printing," in *ASME Proceedings* , Blacksburg, Virginia, 2016.

[65] J. Tarbutton, T. Kurfess, T. Tucker and D. Konobrotskyi, "Gouge-free voxel-based machining for parallel processors," *The International Journal of Advanced Manufacturing Technology,* vol. 69, 2013.

[66] P. Black, Dictionary of Algorithms and Data Structures, 2005: U.S. National Institute of Standards and Technology (NIST), p. 329.

[67] C. Leiserson, C. Stein and T. Cormen, Introduction to Algorithms, MIT Press, 1990.

[68] "Introduction to Bioalgorithms, Molecular Sequence Analysis," University of California, San Diego.

[69] AWS, "What is AWS?," Amazon, [Online]. Available: https://aws.amazon.com/what-is-aws/. [Accessed 5 7 2017].

[70] E. Oberg, F. Jones and H. Holbrook, Machinery's Handbook 30th Edition, 2016.